JASMIN

Le T.DOS

ET SES FICHIERS pour ORIC 1 & ATMOS

P.M. BEAUFILS
C. ARNAUD

JASMIN

Le T.DOS

ET SES FICHIERS pour ORIC 1 & ATMOS

P.M. BEAUFILS
C. ARNAUD

INTRODUCTION

Un système Informatique sans lecteur de disquettes n'est pas complet et ne permet pas d'effectuer des travaux professionnels. Mais un lecteur de disquettes est aussi insuffisant : pour qu'il soit performant, il faut qu'il soit accompagné d'un système d'exploitation (DOS) digne de ce nom. Aujourd'hui, c'est la réalité pour l'ORIC ATMOS, grâce à Jasmin et à son TDOS. Plus de 40 instructions faciles d'emploi, complétant le BASIC de l'unité centrale et assurant une gestion efficace et sophistiquée des fichiers à accès séquentiel et à accès direct. Ces performances ouvrent le champ aux utilisateurs professionnels qui, pour le prix modique de l'ensemble, auront entre les mains un magnifique outil de travail.

L'amateur donnera aussi également une nouvelle dimension à son système micro-informatique.

QUESTIONNAIRE

(à retourner aux Etablissements TRAN)

1 Ce livre vous a-t-il satisfait ? OUI NON

2 Le trouvez-vous complet ? OUI NON

3 Donnez une note (sur 10) à chacun des chapitres :

	Note
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

4 Trouvez-vous qu'il y a :

☐ trop de programmes
☐ pas assez de programmes

5 Quels chapitres auriez-vous aimé voir plus développés ? . . . . . .

6 Y a-t'il assez d'exemples :

☐ OUI
☐ NON

Toujours soucieuse de la qualité de ses produits, et de la satisfaction de ses clients, la Société TRAN fournira gracieusement aux utilisateurs toute nouvelle version du TDOS. (disquette non fournie).

Enfin, et comme nous l'avons dit, un DOS est un instrument scientifique de gestion ; la protection contre les erreurs n'est pas aussi raffinée que pour un BASIC. Aussi, est-il fortement recommandé de lire ce livre avec attention : un DOS n'est pas un jeu. Il vous sera facile de « coincer » le TDOS, au risque de détruire vos fichiers, mais où est l'intérêt ?

Pour terminer, rappelons qu'il est d'usage de toujours conserver un double de chaque disquette...

Bon travail avec Jasmin.

I - GENERALITES
1° - la disquette : aspects physique et logique 7
2° - mise en route de Jasmin 12
3° - l'appel du DOS 18
⊗ l'adresse 2F5
⊗ la syntaxe
⊗ la description syntaxique

II - A LA DECOUVERTE DU TDOS
1° - formatter une disquette 21
2° - monter un second lecteur ! MOUNT «2» - ! MASTER «—» . . 27
3° - START et UNSTART 28
4° - les différents type de fichier 29
5° - la copie de fichiers 31

III - TDOS ET BASIC (1)
⊗ ! SAVE 35
⊗ ! SAVE 37
⊗ ! LOAD 36
⊗ ! RENAME 37
⊗ ! DEL 38
⊗ ! LOCK 38
⊗ ! UNLOCK 38
⊗ ! COPY 39
⊗ ! CUT 39
⊗ ! MERGE 40

IV - LES FICHIERS DE DONNEES
1° - les matrices ! MSAVE et ! MLOAD 46
2° - les écrans haute et basse résolution 54
3° - autres zones mémoires 55

V - LES INSTRUCTIONS DE TRAITEMENT DE FICHIER
1° - Qu'est-ce qu'un fichier ? 56
2° - Quel type de fichier ? 57

VI - MANIPULATIONS SUR LES FICHIERS A ACCES SEQUENTIEL
1° - Instructions de traitements dynamiques de fichiers à accès séquentiel 60
2° - Exemple 62
3° - Examen du fichier 69

3° - Recopie du fichier 72
3° - Extension du fichier 75
3° - Suppression d'un enregistrement 78
3° - Classement d'un fichier 79
4° - Traitement d'erreurs 85
5° - Première application 92

VII - EXEMPLE D'UTILISATION DES FICHIERS
A ACCES SEQUENTIEL 96

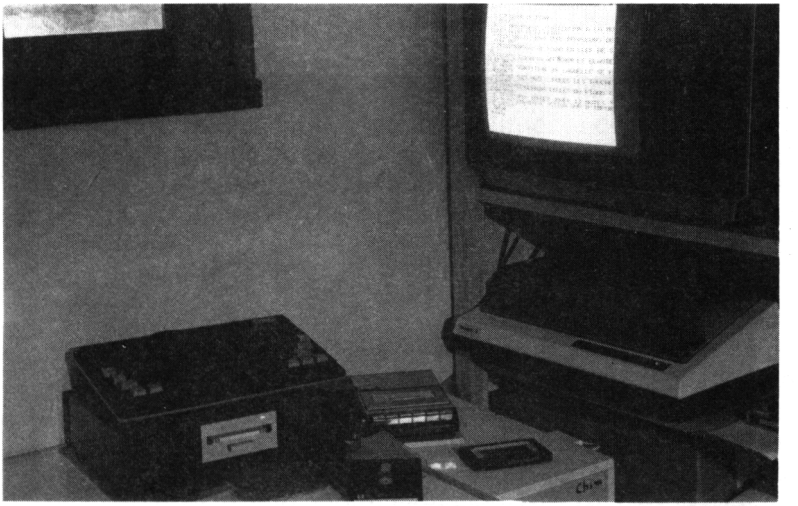
VIII - FICHIERS A ACCES DIRECT INTRODUCTION
1° - Les instructions spécifiques 112
2° - Remplir un fichier 117
3° - Lecture d'une fiche 120
4° - Insertion d'une fiche 121
5° - Suppression d'une fiche 124
6° - Liste séquentielle 126
7° - Recherche par clé simplifiée 128
8° - Exemple concret d'application 130

IX - ALGORITHMES DE TRI
1° - Ripple 151
2° - Bulle 152
3° - Shell 153
4° - Shell-Metzner 155
5° - Quicksort 156
6° - Trisalphanumériques 162

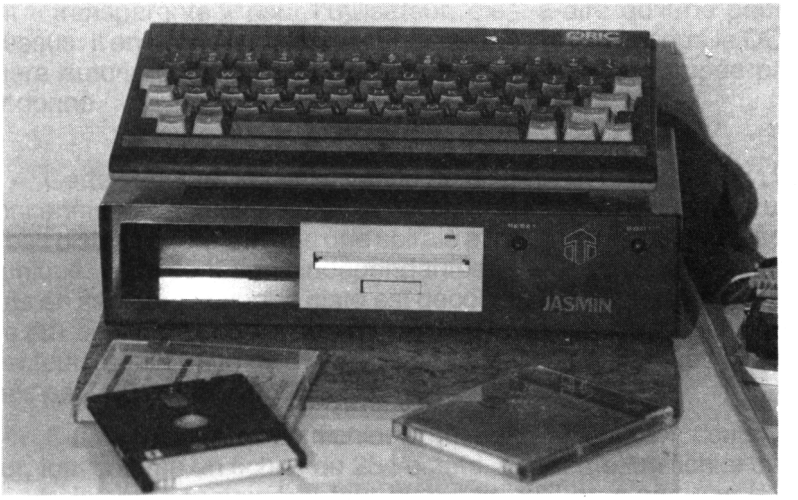
X - FICHIERS A ACCES DIRECT : PROGRAMMATION AVANCEE
1° - Avec constitution de la table d'index en début de session . 164
2° - Avec sauvegarde de la table d'index 168
3° - Clé calculée (HASH-CODE) 176

XI - LANGAGE MACHINE ET TDOS 184

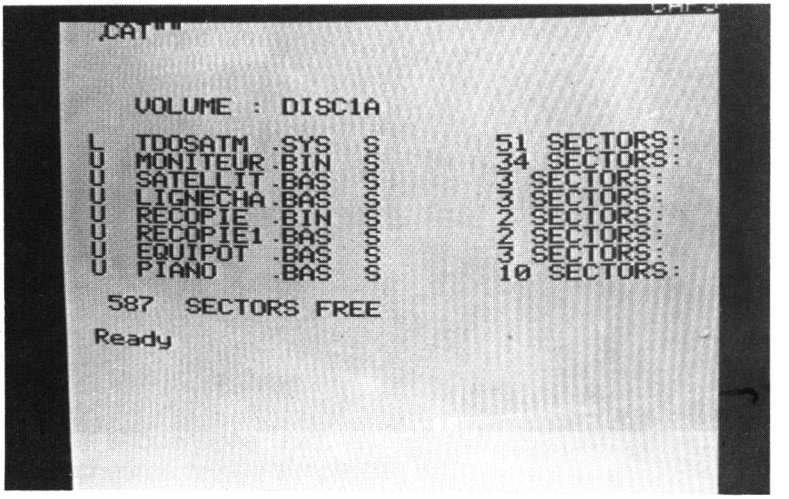
XII - REPERTOIRE DES INSTRUCTIONS DU TDOS 200



Un système informatique « grand public » complet



Le lecteur de disquettes Jasmin



Le « catalogue » d'une face de disquette

1° - LA DISQUETTE : ASPECTS PHYSIQUE ET LOGIQUE

Une disquette se présente sous la forme d'un petit disque, tout comme nos 33 tours familiers. Elle peut être « écrite » sur ses 2 faces. Pour les lecteurs les plus simples, il faut retourner la disquette pour pouvoir accéder à la totalité de sa capacité. On distingue, par convention, une face A et une face B, qui sont évidemment absolument identiques. Le lecteur de disquettes Jasmin offre pour cela une commodité : une petite LED, située sur la face avant du lecteur, s'éclaire au vert pour la face A et en rouge pour la face B.

Une disquette vierge est absolument inutilisable telle quelle. En effet, pour profiter au maximum des possibilités (rapidité, fiabilité) d'une telle mémoire de masse, il faut, comme pour une carte d'Etat Major, « quadriller » la disquette afin que le DOS — logiciel d'exploitation du système — puisse stocker et retrouver quasi instantanément et de façon sûre une information. Celle-ci se présente de façon classique : un octet = groupe de 8 bits valant 0 ou 1 = une information élémentaire. Ces informations, toujours par soucis d'efficacité, sont groupées sous le nom général de fichier, un fichier pouvant être un programme basic, un programme en langage machine, ou plus généralement un ensemble d'informations liées par un point commun. On distingue ces fichiers par leur nom, absolument arbitraire, et par leur type (basic, binaire, data...).

Revenons à la structure de la disquette. Cette structure est absolument « transparente » pour l'utilisateur, c'est-à-dire qu'il ne peut pas agir dessus. Il en aura simplement connaissance par le fait que le DOS lui indiquera automatiquement, sur sa demande, la place occupée par un fichier donné.

Cette structure est la suivante : le disque est découpé en 41 pistes concentriques (circulaires) sur chaque face. Contrairement à un 33 tours, ces pistes ne forment pas une spirale et donc ne constituent pas une spire unique. Elles sont donc indépendantes. A leur tour, ces pistes sont divisées en secteurs : chaque piste est découpée en 17 secteurs, chacun d'entre eux occupant donc un arc de cercle d'ouverture 360° : 17 = 21,2°. Ces secteurs sont l'unité de stockage de base : ils sont prévus pour être occupés par 256 octets, soit 256 x 8 = 2048 bits.

Cela veut dire qu'un fichier contenant 12 octets, occupera un secteur, tout comme un fichier de 254 octets. De même, un fichier de 300 octets emploiera 2 secteurs. Il faut bien voir qu'il ne s'agit pas là d'un « standard », mais d'un compromis effectué par le concepteur du DOS. Il aurait été possible de concevoir des secteurs de 1024 octets par exemple,

mais alors les trois fichiers précédents auraient occupé un secteur chacun, d'où une perte sèche, au point de vue stockage de 3 x 1024 — (12 + 254 + 300) = 2504 octets, au lieu de 4 x 256 — (12+ 254 + 300) = 456 octets.

Mais — nous l'avons déjà dit — la connaissance de ce « quadrillage » n'est pas nécessaire pour utiliser un lecteur de disquettes. Nous allons donc distinguer la disquette « physique », qui est telle que nous l'avons décrite, de la disquette « logique » dont nous aurons seule la connaissance. Celle-ci est définie de la manière suivante :

⊗ à chaque unité de lecteur, sera attribué un **numéro d'unité,** qui le distinguera des autres,

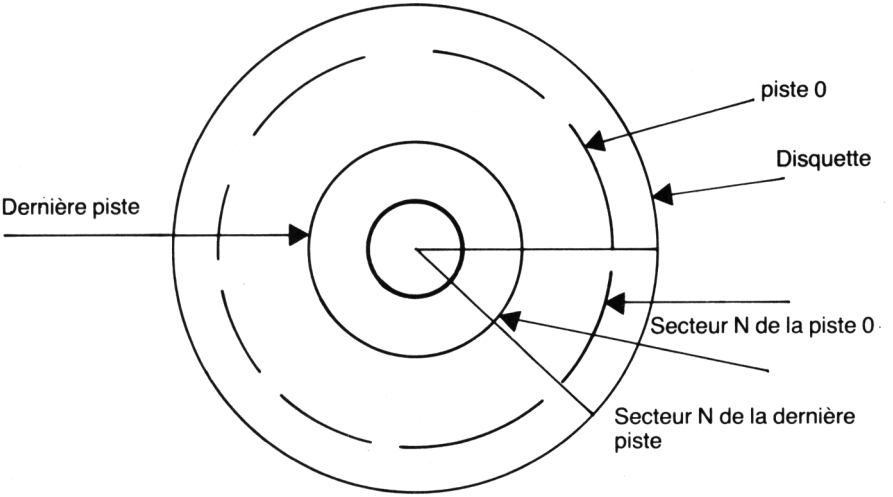
⊗ à chaque disquette, sera attribuée un nom, appelé **nom de volume,**

⊗ à chaque fichier d'une disquette, sera attribué un **nom de fichier** et un **type** qui indiquera quel est le type des informations stockées sous ce nom : programme basic, en langage machine...

L'exploitation d'un fichier donné se fera alors de la façon suivante :

⊗ choisir la disquette convenable,
⊗ choisir le lecteur qui sera exploité,
⊗ choisir le fichier auquel on veut accéder.

Le système est alors complètement défini et il n'y a ambiguité ni pour le DOS, ni pour l'utilisateur. Il est alors commode d'affecter, un numéro, appelé **numéro logique du fichier** qui permet de condenser l'ensemble de ces informations. C'est là l'aspect « disquette logique », par lequel il peut y avoir effectivement dialogue entre unité centrale du micro-ordinateur et sa mémoire (externe) de masse. Le DOS est le logiciel qui permet ce dialogue, mais qui, de plus, intègre un certain nombre de commandes fort proches du BASIC au point de vue syntaxe ; celles-ci assurent l'exploitation de toutes les informations présentes sur la disquette. Un lecteur de disquettes est donc à ce point de vue autre chose qu'un magnétophone. Ce dernier ne permet que le stockage et la récupération d'information ; il ne peut pas les rechercher, ni les modifier directement sur la bande. On peut même estimer que ces deux périphériques sont complémentaires.

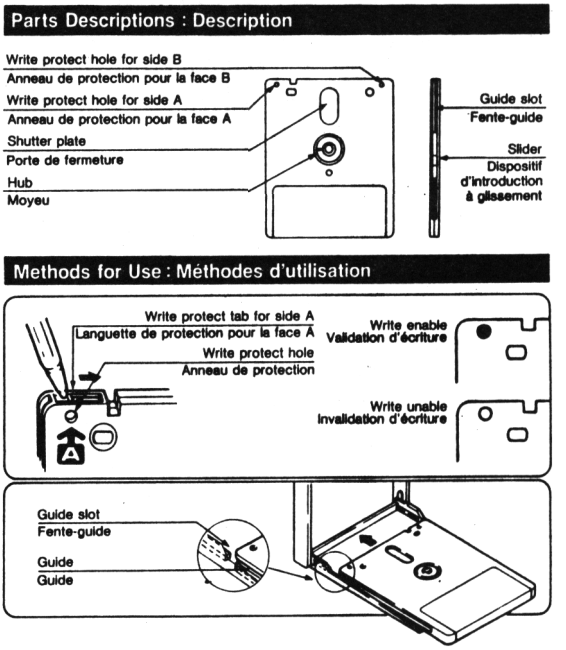


une disquette formatée
il y a 41 pistes par face ; chaque piste est divisée en 17 secteurs.
Chaque secteur contient 256 octets.
Cela fait un total de 176 koctets par face.

Une disquette 3 pouces possède une protection, qui permet d'empêcher qu'on écrive dessus, tout en en permettant la lecture (tout comme pour une cassette audio — mais là la protection est définitive puisqu'il faut casser une languette — ou une cassette vidéo). A l'origine, un petit trou, marqué « A » ou « B » suivant la face, est fermé par un index rouge. Devant ce trou, sur la tranche de la disquette, se trouve une languette située dans la partie gauche d'une fente. La pointe d'un stylo bille permet de déplacer cette languette vers la droite et de protéger ainsi la disquette contre tout risque d'effacement accidentel. L'index rouge disparait alors et le trou est libéré. Cette opération est bien sûr reversible : le retour de la languette à sa position initiale (vers la gauche) permet de nouveau d'écrire sur la face correspondante de la disquette.

Remarquez sur les côtés du boîtier des fentes permettant le guidage de la cassette dans son boîtier. Il est curieux de constater que cette protection apparemment mécanique soit prise en compte par l'électronique.

Il faut bien sûr ne pas déplacer un lecteur Jasmin, contenant une disquette. De préférence, celui-ci doit être enlevé avant la mise hors tension de l'appareil.



Unheard of High Performance Realized with Ultra-Miniature Floppy Disk Drive

HITACHI's advanced electronics technology has made the floppy disk so small in size and so high in performance. This newest version is about 1/5 of the size and 1/2 of the weight of the conventional 5-inch floppy disk drive hitherto available. It has the drive capability of 500 Kbytes per disk (both sides A and B usable), which is significantly higher than that of the mini-floppy disk drive.

■ Features

- Plug compatible with mini-floppy units.
- One-touch loading/ejecting mechanism is adopted for insertion and ejection of the disk merely by single hand based on the one touch method.
- Long-life and high-performance brushless DD (Direct Drive) motor is mounted on board.
- High-speed access (55m sec on an average) steel band mechanism is in use.
- HFD 305S disk unit (optional) is so compact that it can readily be put in the bust pocket.

■ SPECIFICATIONS

Item		Unit	Single-sided density recording			Double-sided density recording		
	Unformatted							
	Disk capacity (double sided)	Bytes	250K			500K		
Storage capacity	Surface capacity	Bytes	125K			250K		
	Track capacity	Bytes	3125			6250		
	Formatted							
	Byte/sector		128	256	512	256	512	1024
	Sector/track		16	9	5	16	9	5
	Disk capacity (double sided)	Bytes	164K	184K	205K	328K	369K	410K
	Surface capacity	Bytes	82K	92K	102K	164K	184K	205K
	Track capacity	Bytes	2048	2304	3560	4096	4608	5120
Disk	No. of memory blocks		2			2		
	No. of cylinders		40			40		
	Total No. of tracks		80			80		
Recording system	System		FM			MFM		
	Max. recording density	BPI	4473			8946		
	Track density	TPI	100			100		
	Data transfer speed	bits/sec	125K			250K		
Process time	Average access time	msec	55					
	Between tracks	msec	3					
	Settling time	msec	15					
	Average rotation stand-by time	msec	100					
	Motor start time	sec	1 max					
Media RPM		rpm	300					
Dimensions & Weight	W × H × D	mm	90 × 40 × 150					
	Weight	kg	0.7					
Power source	DC 12V		12V ±5% 1.0A max					
	DC 5V		5V ±5% 0.8A max					
Media used			3" compact floppy disk					
Drive interface			Plug compatible with 5.25" FDD					

2° - MISE EN ROUTE DE JASMIN

Vous disposez donc d'un lecteur de disquettes Jasmin et d'un micro-ordinateur adapté. La liaison entre ces deux éléments est simple : un cable plat (dit en nappe) muni d'un connecteur. Il n'y a pas de possibilités d'erreur de branchement : le connecteur est muni d'un détrompeur, éliminant toute ambiguité. Ce cable plat apporte également la tension de + 5V nécessaire au fonctionnement de l'unité centrale. Il faut donc **supprimer** l'alimentation fournie avec celle-ci. L'ordre des opérations est le suivant :

⊗ Jasmin étant **débranché** (si vous avez un doute sur la position de l'interrupteur) du secteur, connectez le cable plat au micro-ordinateur,

⊗ connectez Jasmin au secteur, puis commutez son bouton de mise en marche. Nous supposons alors le moniteur branché (comme d'habitude),

⊗ une magnifique image apparaît sur l'écran (je vous mets au défi de réaliser la même par programme !) : lignes horizontales de toutes couleurs, clignotements... Au bout de quelques secondes, apparaît l'image classique :

```
ORIC EXTENDED BASIC V1.1
© 1983 TANGERINE
xxxxx BYTES FREE
READY
```

⊗ si cela ne s'est pas produit, il faut vous munir d'un crayon ou d'un stylo bille (le même que celui qui vous sert d'habitude à faire le RESET sous l'ORIC), et appuyer sur le bouton R ou RESET du Jasmin.

ATTENTION : bien que portant le même nom, ces deux « RESETS » n'ont pas la même fonction. Celui de l'ORIC est dit « à chaud » et permet de récupérer la main sans perdre les informations présentes en RAM (les programmes...). Par contre, celui de Jasmin est un RESET « à froid » et a les mêmes conséquences qu'une mise sous tension d'ORIC : perte de tous les programmes présents en RAM. Ce bouton, par la suite, ne doit donc être utilisé qu'en dernière extrémité. Remarquons que « BOOT » produit également un RESET à chaud et permet dans certaines circonstances de récupérer la main.

⊗ Cette fois-ci, l'image précédente doit apparaître. Sinon, éteignez Jasmin et recommencez la procédure.

⊗ Introduisez alors la disquette maîtresse fournie avec Jasmin.

⊗ Appuyez sur le bouton « B » ou « bootstrapping ». La disquette se met en rotation pendant quelques secondes, puis vous obtenez sur l'écran :

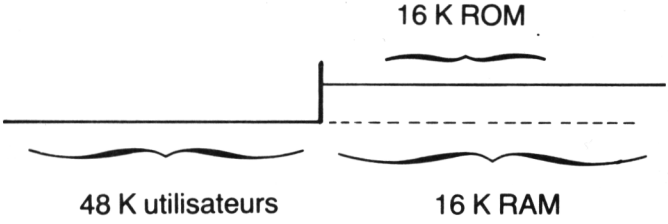
TRAN DISK OPERATING SYSTEM VF1,0

© 1983 TECHNOLOGIE RECHERCHE

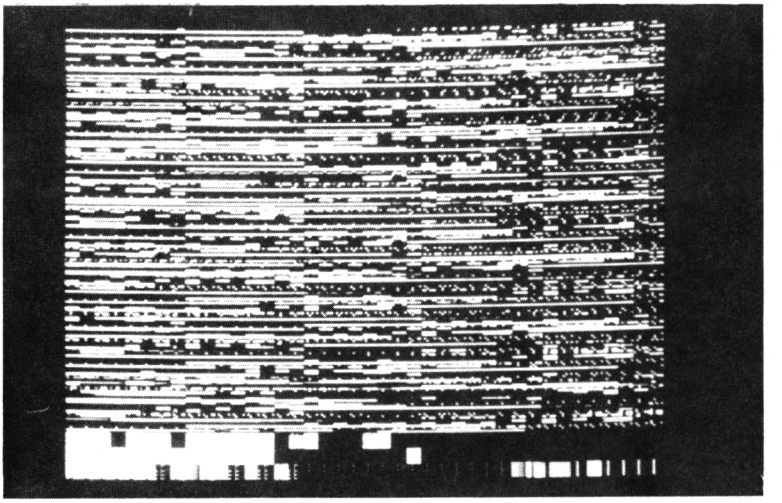
& APPLICATIONS NOUVELLES. READY

Il s'est alors passé la chose suivante :

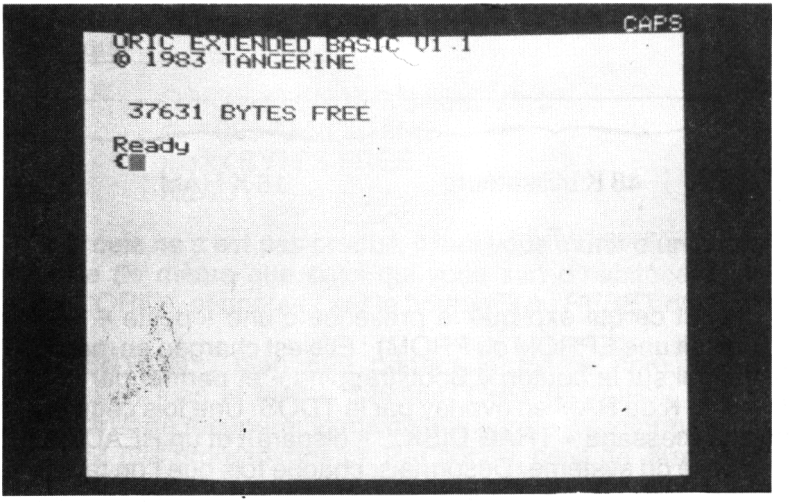
le DOS du Jasmin occupe la RAM en overlay d'ORIC. Que cela signifie-t-il ? La mémoire d'ORIC est de 48 K (utilisateurs, mais en fait les mémoires d'écran occupent une partie de cet espace), plus 16 K de ROM contenant le BASIC. Mais en fait, ces 16 K recouvrent 16 K de RAM qu'il est possible d'exploiter, si l'on n'utilise pas le BASIC.



C'est ce qui explique la présence d'une « petite » ROM dans Jasmin (en fait une EPROM ou PROM) : Elle est chargée en mémoire vive lors de l'appui sur le bouton « Bootstrapping » et permet par la suite de remplir les 16 K de RAM en overlay par le TDOS. Une fois cette opération terminée, le message « TRAN DISK... » disparaît et un READY annonce la disponibilité du système. Désormais, chaque fois que l'on tapera ! suivi d'une commande appropriée et syntaxiquement correcte on aura accès au DOS, c'est-à-dire la possibilité de dialoguer avec le lecteur.

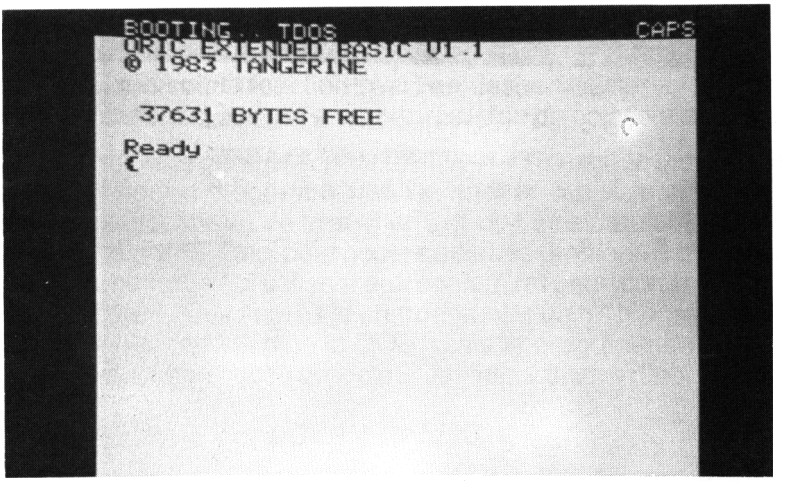


A la mise sous tension, apparaît un curieux dessin... Celui-ci disparaît spontanément, ou bien après l'action sur le RESET Jasmin.

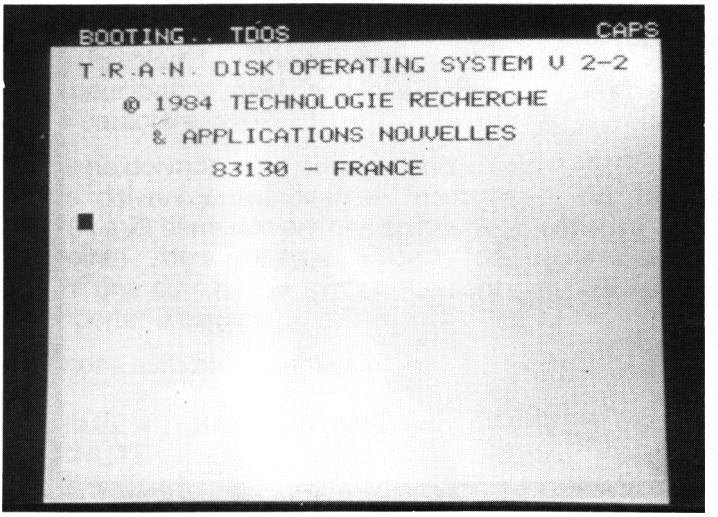


Et voilà ! Tout est O.K.

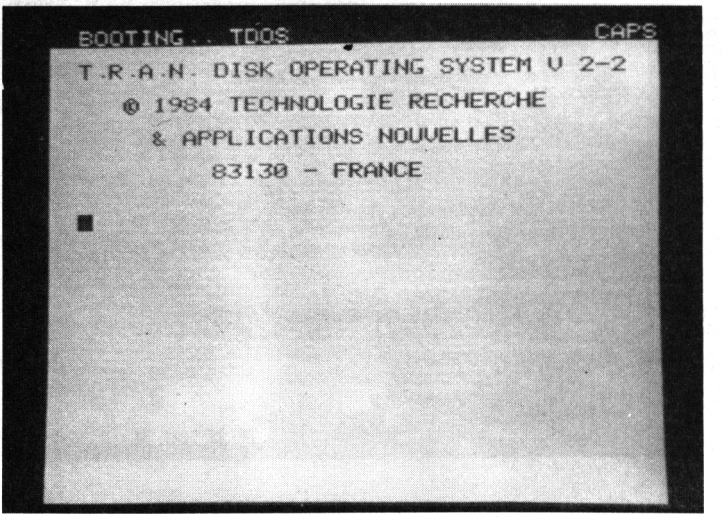
Introduisez une disquette « maîtresse », porteuse du TDOS et appuyons sur le bouton « BOOT » de Jasmin.

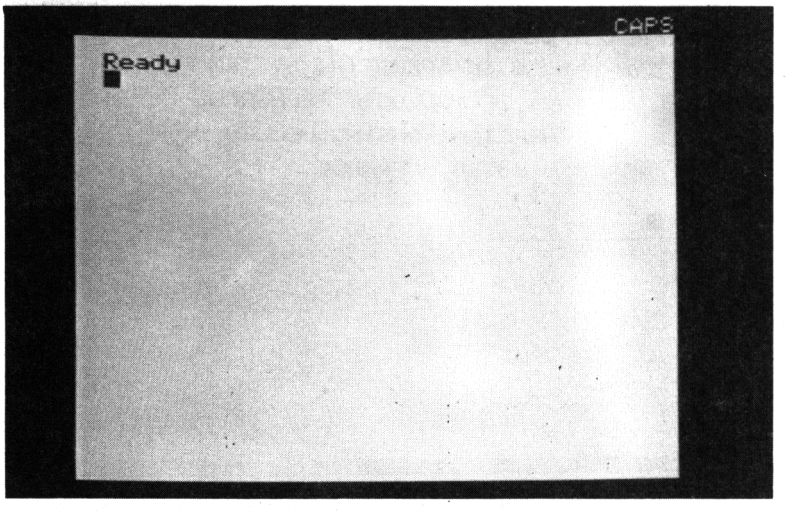


Le Bootstrapping s'effectue correctement et dure quelques secondes.



S'affiche alors le logo « TRAN », la RAM en overlay est alors chargée par le TDOS.





Opération terminée. ORIC est prêt, le TDOS est présent.

3° - L'APPEL DU DOS

⊗ l'adresse # 2F5

Quelque soit la fonction que l'on désire réaliser avec le lecteur Jasmin, il est essentiel que celle-ci soit précédée du point d'exclamation : !

De manière générale, pour ORIC, le « ! » est une instruction permettant d'appeler un programme (ou routine) en langage machine, dont l'adresse (hexadécimale) de branchement doit être préalablement chargée à l'adresse # 2F5. Ceci peut poser quelques problèmes avec certains logiciels du commerce qui utilisent eux également cette commande. Il faudra donc procéder ainsi : après le chargement d'un tel programme, le fait d'utiliser « ! » n'appellera plus le DOS, mais le programme en question. Pour avoir de nouveau accès au lecteur Jasmin, il faudra d'abord effectuer la commande :

DOKE # 2F5, 1024

On pourra d'abord relever — si toutefois cela est possible — le nombre présent en # 2F5 par :

PRINT DEEK (# 2F5)

de manière à pouvoir par la suite le repoker en # 2F5 et retrouver ainsi la possibilité d'utiliser le programme.

LA SYNTAXE :

La plupart des commandes exigent de fournir un nom : noms de fichiers, noms de volume,... Ces noms peuvent cependant être des variables chaînes. Leur longueur est dans tous les cas tronquée à droite, à partir du huitième caractère.

⊗ Ils ne doivent contenir aucun mot clé du BASIC.

⊗ Ils doivent obligatoirement commencer par un caractère alphabétique (A à Z) et ne doivent pas comporter d'espaces (code ASCII = 32), ni de point : . , de signe égal : =, de ' ni de virgule,.

Enfin, une commande syntaxiquement groupée ne doit également pas comporter d'espace.

Donnons quelques exemples :

— A$ = « CHIMIE » : ! SAVE « CHIMIE » ..... oui
— ! SAVE « CHUTE » ..... oui
— ! SAVE « PIN UP » ..... non, car il y a un espace
— ! LOAD « PRINT » ..... non, car il y a un mot clé du BASIC
— ! RENAME « Z1234,5 » ..... non, car il y a une virgule
— ! « PUF123 » ..... oui
— ! SAVE « 1.A$.BIN » ..... oui

De manière plus générale, la totalité de ce qui se trouve entre guillemets dans une commande peut être fournie sous la forme d'une variable de chaîne de caractères.

LA DESCRIPTION SYNTAXIQUE :

Pour chaque commande sera décrite un modèle indiquant les différentes quantités intervenant dans celle-ci et la syntaxe à utiliser. Nous utiliserons le symbolisme suivant :

⊗ les parties d'une commande situées entre parenthèses sont optionnelles et leur absence amène le DOS à prendre une valeur par défaut. Celle-ci est toujours clairement indiquée dans la description de la commande.

Exemple : ! CAT « (numéro de lecteur.) »
signifie que ! CAT «» donnera le catalogue de la disquette présente dans le lecteur n° 1 (ou le seul s'il n'y en a qu'un) alors que ! CAT « 2. » donnera le contenu de la disquette située dans le lecteur n° 2.

⊗ Les parties soulignées par un trait continu sont obligatoires et ne peuvent être omises.

Exemple : pour un programme BASIC s'appelant CHIMIE, on pourrait décrire le formalisme de l'instruction SAVE par :
! SAVE « (numéro de disque.) **NOM PROGRAMME** »
Ainsi, les deux expressions suivantes sont correctes :
! SAVE « 1.CHIMIE » (équivalent à ! SAVE « CHIMIE ») et
! SAVE « 2.CHIMIE »

⊗ Les parties soulignées par un trait discontinu sont obligatoires de façon conditionnelle. La condition de présence obligatoire est toujours indiquée clairement dans le texte décrivant la commande.

Exemple : pour sauvegarder un programme en langage machine ou en BASIC, on pourrait décrire la syntaxe commune de la façon suivante :
! SAVE « (numéro de disque.) NOM PROGRAMME (.TYP) (,AD,AF) »
Cela signifie que, pour un programme en langage machine, il est obligatoirement de fournir l'adresse AD de la première instruction de celui-ci et l'adresse AF de la dernière instruction. Dans un tel cas, (.TYP) sera remplacé par .BIN (BIN pour binaire).

Ainsi :

! SAVE « CHIMIE » et ! SAVE « 2.CHIMIE » sont corrects pour un programme BASIC, alors que :

! SAVE « MONITEUR.BIN, # 9000, # 10000 »

! SAVE « 2.MONITEUR.BIN,X,Y » (X et Y ayant été préalablement définis) sont des expressions correctes pour un programme en langage machine.

Par contre :

! SAVE « MONITEUR, # 9000,Y » ne l'est pas, car il manque l'indicateur de type (.BIN).

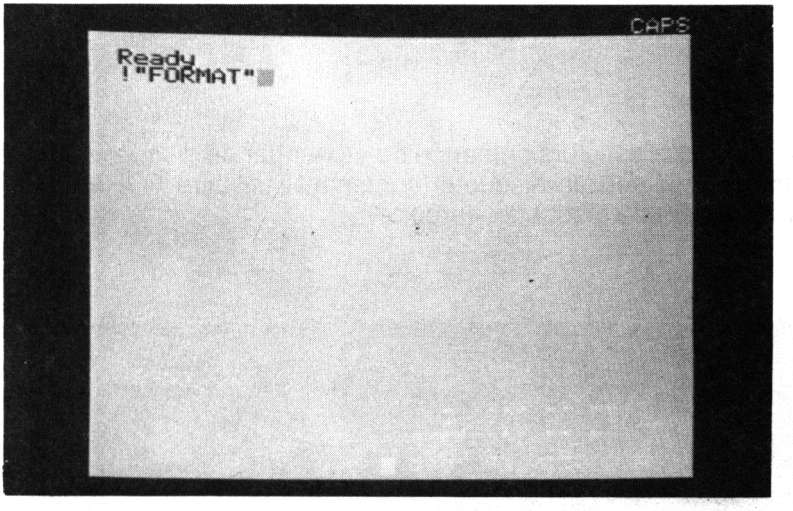
A LA DECOUVERTE DU TDOS

1° - FORMATTER UNE DISQUETTE

Nous avons décrit précédemment l'aspect physique d'une disquette ; mais une disquette vierge (sortie de sa boîte et de son cellophane) ne présente pas cet aspect. Il n'y a ni pistes, ni secteurs...

Le premier rôle du DOS est donc de la rendre capable de stocker des informations. Cette opération s'appelle **le formattage.** Il n'y a bien sûr aucun indice physique permettant de séparer les pistes et les secteurs. Ce sont donc des informations binaires, inscrites par le DOS sur la disquette, qui délimiteront les frontières entre les différentes unités de mémorisation de base. D'autre part, le nom de la disquette, puis plus tard les noms de fichier, leur longueur, leur type seront également inscrits sur celles-ci. Il faut donc bien voir que cette opération de formattage va avoir pour conséquence d'utiliser une certaine place sur la disquette. Mais ceci, l'utilisateur ne le sait pas : l'opération est transparente pour lui. C'est pour cela que la structure physique de la disquette que nous avons décrite est celle d'une disquette formattée, c'est-à-dire telle qu'elle se présente dans son état « logique ». Il n'est en fait pas impossible qu'il y ait en réalité 44 pistes, dont 3 correspondraient aux paramètres de formattage et ne seraient donc pas exploitables, si ce n'est par le DOS, qui s'en servirait à l'insue de l'unité centrale. Cette opération de formattage s'effectue en 3 temps. Il est d'abord essentiel de posséder une disquette dite **maîtresse** possédant d'une part le DOS, d'autre part un programme utilitaire appelé « FORMAT ». Ce devrait être votre cas... suivez alors le film des événements :

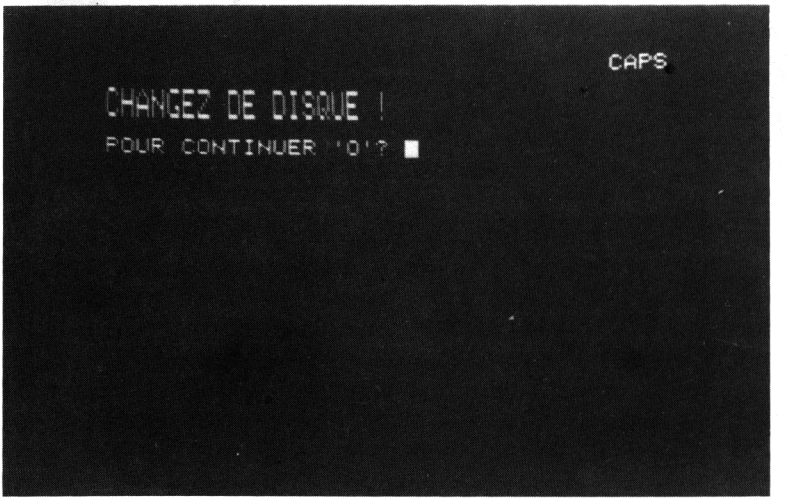
Employons une disquette contenant l'utilitaire « FORMAT ». Attention, il s'agit-là d'un programme (BASIC) distinct du TDOS et est donc lancé comme un programme quelconque par l'expression : ! « FORMAT » (et RETURN).

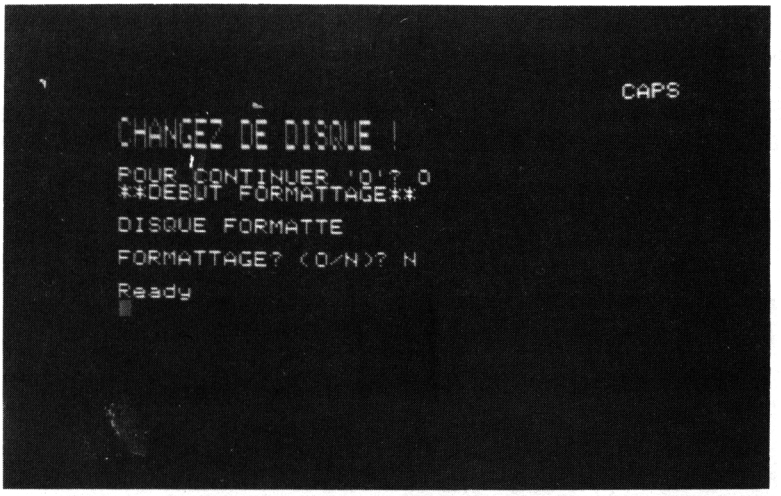


La préparation à l'opération de formattage s'effectue alors :

...puis se termine :

Il vous est alors demandé de « changer de disque », c'est-à-dire d'extraire du lecteur la disquette contenant l'utilitaire FORMAT et de la remplacer par une disquette vierge :





Appuyez sur « O » : le formattage s'effectue. Il vous est alors demandé si vous voulez procéder à un nouveau formattage ; c'est certainement votre cas, puisqu'il reste la seconde face (B ou voyant rouge), vierge, de la même disquette. Extrayez donc celle-ci, retournez-la et introduisez-la de nouveau dans le lecteur. Le voyant rouge s'allume. Appuyez sur « O ». Dès que « Ready » apparaît, l'opération est terminée. Vous répondez alors « N » (NON) à la nouvelle demande de formattage.

La mise en œuvre de cette disquette n'est pas terminée pour autant. Il vous faut en effet maintenant initialiser les deux faces de la disquette, c'est-à-dire en particulier leur donner un nom. Nous avons un seul lecteur, il s'agit de notre troisième disquette. Nous tapons alors :

! INIT « 1.DISK 3 B »

premier lecteur — nom donné à la face considérée

et Return.

Nous retournons la disquette, et tapons :

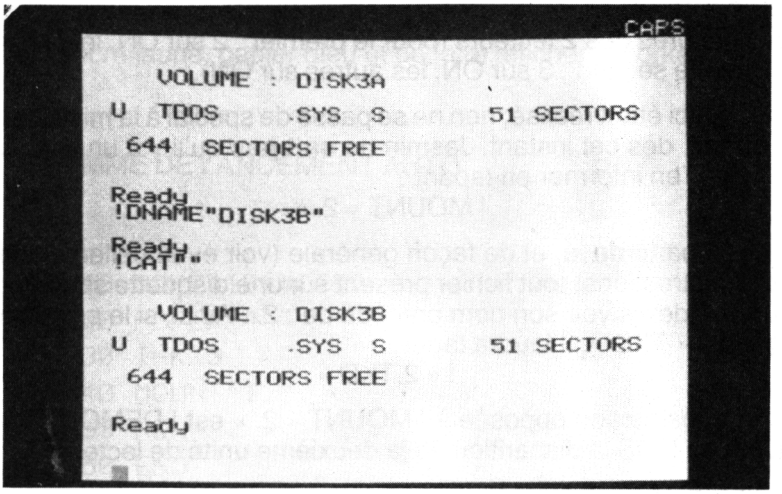
! INIT « 1.DISK3A » (et Return)

Notre disquette est prête (enfin !) à recevoir des programmes. Il est judicieux alors de la rendre « maîtresse » sur ses 2 faces, c'est-à-dire de la rendre porteuse à son tour du TDOS. C'est très simple : tapons : ! MASTER « 1.TDOS » (et Return) puis après un retournement de la disquette, recommençons. La mise en route de la disquette est terminée.

```
                                        CAPS
!INIT"1.DISK3B"
Ready
!INIT"1.DISK3A"
Ready
!MASTER"1.TDOS"
Ready
!MASTER"1.TDOS"
Ready
```

Nous pouvons alors demander le « catalogue » de la disquette. Tapons : ! CAT «» (et Return). Le nom de volume s'affiche (DISK3A), puis le (seul) programme : TDOS ; les indications U (fichier non protégé : Unlock), SYS (système d'exploitation), S (séquentiel), '62 SECTORS s'affichent également, ainsi que le nombre de secteurs libres.

Pour terminer ce paragraphe, supposons que le nom choisi pour cette face ne convienne pas (ou plus). Tapons simplement : ! DNAME « DISK3B » (et Return). Une nouvelle demande de catalogue nous permet de vérifier que le nom de volume a bien été changé.



Pour terminer, signalons que toutes ces opérations (formattage, initialisation, master) seront à effectuer si vous désirez effacer une disquette ancienne.

2° - MONTAGE D'UN SECOND LECTEUR :

Vous avez acquis dans un premier temps un lecteur Jasmin ; vous désirez maintenant en acquérir un second. Cela est possible : l'alimentation du Jasmin est capable d'alimenter deux lecteurs et l'ordinateur lui-même.

La société TRAN vous livrera alors un « kit » comprenant :
- le deuxième lecteur,
- un cable d'alimentation +12,+5,0V, à connecter comme celui du premier lecteur, sur le contrôleur (des trous sont prévus),
- un connecteur destiné au cable plat de liaison entre contrôleur et lecteurs. Les deux lecteurs doivent être branchés en parallèle (attention au sens).

Une notice jointe au kit vous fournira tous les renseignements complémentaires nécessaires. Faire attention à la position des commutateurs miniatures des 2 lecteurs (pour le premier : 2 sur ON, les autres sur OFF ; pour le second : 3 sur ON, les autres sur ON).

Ceci étant réalisé, rien ne se passe de spécial à la mise sous tension. En fait, dès cet instant, Jasmin **ne sait pas** qu'il y a un second lecteur ; il faut l'en informer en tapant :

! MOUNT « 2. ».

A partir de là, et de façon générale (voir éventuellement la syntaxe des instructions) tout fichier présent sur une disquette située dans le lecteur n° 2 devra voir son nom précédé de : 2. . Ainsi, si le programme à charger est « TKD », il faudra taper :

! « 2.TKD »

L'instruction opposée à ! MOUNT « 2. » est ! DEMOUNT « 2. », indiquant au DOS la disparition de la deuxième unité de lecteur.

3° - START ET UNSTART

! Start «»

N'avez-vous jamais regretté que lors de la mise sous tension votre ordinateur choisisse d'écrire en noir sur fond blanc ? Vous aimeriez connaître le catalogue de la disquette au moment où vous avez booté le DOS ?

Vous trouvez fastidieux de devoir indiquer au système et à chaque mise sous tension que vous possédez deux lecteurs ?

Dans ces cas et dans bien d'autres la commande ! START vous sera très utile. Elle permet en effet de lancer automatiquement un programme basic ou en code machine dès que le TDOS est booté.

Le MODE d'emploi de cette instruction est spécial, lisez-le bien.

1° - Recommandation : elle a son importance : la première fois que vous exécutez cette manœuvre, prenez soin de conserver sur une disquette une copie du TDOS. Si le programme qui va être lancé automatiquement se « plante » vous ne pourrez plus récupérer la main sur l'ordinateur (même pas par un reset) et comme chaque fois que le TDOS sera booté le programme sera lancé... vous ne récupèrerez jamais la main. Dans ce cas, la copie vous permettra d'introduire le DOS dans l'ORIC et de supprimer l'effet de ! START«» sur la disquette inutilisable.

2° - Tapez normalement le programme qui doit être lancé, le cas échéant testez-le. Le programme suivant par exemple permet de positionner l'Oric encre jaune, papier bleu et de faire afficher le contenu de la disquette.

A PROGRAMME DE LANCEMENT AUTO

```
10 DOKE#1B,#CCB0
20 PAPER 4
30 INK 3
40 MOUNT"2."
50 NEW
60 END
```

3° - L'instruction 40 est absolument nécessaire

DOKE #1B,#CCB0 pour l'atmos
DOKE #1B,#CBED pour l'oric 1.

A la fin du programme l'oric se branche à l'adresse #1B pour connaître ce qu'il doit faire ; #CCB0 au #CBED sont les adresses de début de la routine qui affiche Ready et rend la main à l'utilisateur.

L'instruction 50 est facultative ; mais elle permet de nettoyer la zone mémoire de programme. Ainsi, on peut commencer à programmer sans s'occuper du programme A.

4° - Sauvez le programme en lui donnant un nom d'**UNE** seule lettre ; par exemple ! SAVE « A » pour un programme écrit en basic, ou ! SAVE « W.BIN » pour un programme en langage machine. (Bien entendu le programme doit être sauvegardé sur la disquette et sur la face sur laquelle porte la commande ! START «»).

5° - Tapez ! START « nom du programme » suivi de RETURN par exemple : ! START « A » ↵ ↵ représente la touche RETURN.

6° - Et enfin tapez ! MASTER « nom du DOS » puis RETURN par exemple : ! MASTER « TDOS » ↵
pour les deux dernières instructions, la syntaxe est la syntaxe habituelle :
! START « (n° disque.) NOM (.TYP) » TYP = BAS ou BIN
! MASTER « (n° disque.) NOM du DOS »
Dorénavant le programme se déroulera à chaque mise en service du système (si disquette qui contient le programme précédemment défini est engagée).

! UNSTART «»

Vous ne désirez plus que le programme soit lancé automatiquement. Il faut simplement utiliser la commande ! UNSTART
! UNSTART « nom du programme qui était lancé » Return
! MASTER « nom du DOS » Return
par exemple : ! UNSTART « A » ↵
! MASTER « TDOS » ↵
La syntaxe complète de ! UNSTART est identique à celle de ! START »
! UNSTART « (n° disque.) NOM (.TYP) »

4° - LES DIFFERENTS TYPES DE FICHIER

Dans le catalogue, obtenu par ! CAT«» (ou bien par ! CAT« (numéro de disque.) », dans le cas de plusieurs lecteurs en ligne), figure, à côté du nom du fichier (ce terme étant pris dans son sens général) le type de celui-ci. Il y en a 8 possibilités.

.BAS : indique que le fichier en question est un programme BASIC, bien que celui-ci puisse comporter un chargement en langage machine. Dans ce cas, les instructions ! SAVE A$, ! LOAD A$, et ! A$ sont opérationnelles, A$ étant le nom du programme, précédé éventuellement du numéro de l'unité de disquette, mais pas obligatoirement du type (ici : .BAS) qui est facultatif dans ce cas. Autres instructions concernées : ! SEARCH, ! RENAME, ! LOCK et ! UNLOCK, ! MERGE, ! DEL, ! COPY.

.SYS : lors du boostrapping d'une disquette vierge, le fait de la rendre « maîtresse », c'est-à-dire porteuse du TDOS la charge d'un programme système gérant toutes les communications avec le lecteur de disquette. Ce programme, et lui seul, est du type SYS. L'utilisateur ne peut créer un fichier quelconque ayant ce type.

.DAT : tous les fichiers définis comme étant du type séquentiel ou à accès direct verront leurs informations stockées sous le type DATA. Cela se fera automatiquement, sans que l'utilisateur ait à le préciser. D'ailleurs, l'indicateur de ce type dans les commandes concernées n'est pas obligatoire, puisqu'il est toujours sous-entendu. Instructions concernées : ! CREATE, ! OPEN.

.BIN (.CMD et .TXT). De manière générale, toutes les informations sous forme binaire présentes dans la mémoire de l'unité centrale seront des fichiers binaires. Ainsi, un programme en langage machine stocké entre les adresses #400 et #4FF pourra être sauvegardé sous le nom choisi, à condition qu'il soit accompagné de la mention .BIN. De même, la mémoire caractères pourra l'être de la même manière. Il y a une différence entre ces deux exemples : le premier est exécutable, puisqu'il s'agit d'un programme, le second ne l'est pas puisqu'il s'agit de données. Ainsi, peut-on distinguer dans le catalogue en remplaçant respectivement .BIN par .CMD et .TXT. Si cela n'a aucune importance pour le DOS, cela permet à l'utilisateur d'avoir plus de souplesse pour le choix du nom des fichiers. Par contre, un fichier binaire A$ par exemple sera exécutable s'il est étiqueté .CMD par : !« — .CMD » alors qu'il ne le sera pas avec la mention .TXT.
Instructions concernées : ! SAVE, ! LOAD, ! SEARCH, ! RENAME, ! LOCK et ! UNLOCK, ! COPY, ! DEL

.ARY : toutes les informations sous forme d'élements de matrice (de tableaux) traitées par les instructions ! MSAVE et ! MLOAD apparaissent dans le catalogue sous le type .ARY (ARRAY en anglais). De tels fichiers ne sont pas manipulables.
Instructions concernées : ! MSAVE, ! MLOAD, ! COPY, ! SEARCH, ! RENAME, ! LOCK et ! UNLOCK, ! DEL

.SCR : Toutes les informations relatives à des contenus d'écran (haute et basse résolution), traitées par les instructions ! LSCR et ! HSCR sont sauvegardées sur la disquette à l'aide du type .SCR (SCREEN en anglais). Comme les précédents, ces fichiers ne sont évidemment pas exécutables.
Instructions concernées : ! LSCR, ! MSCR, ! COPY, ! SEARCH, ! RENAME, ! LOCK et ! UNLOCK, ! DEL

En résumé :

		Basic	Binaire			System
	fichiers		données	écran	matrice	
exécutables		.BAS	.BIN .CMD			.SYS
non exécutables	.DAT		.TXT	.SCR	.ARY	

5° - LA COPIE DE FICHIERS :

Il peut être utile de dupliquer un fichier et cela, pour plusieurs raisons.

Il est sage de toujours conserver en double exemplaire, et sur deux disquettes différentes, un programme d'une certaine valeur. Une mauvaise manipulation pourrait en effet faire perdre définitivement l'enregistrement de ce programme. On conservera donc en général un « original » et l'on se servira en temps normal de la « copie ». D'autre part, pour tous ceux d'entre les lecteurs qui viennent d'acquérir un Jasmin, leur problème va être de passer sur disquette tous leurs anciens enregistrements sur cassette. Malheureusement, comme nous le verrons, cela posera quelques problèmes.

Enfin, il y aura nécessité de copier des fichiers lorsque des possesseurs d'ORIC voudront échanger entre eux des logiciels.

Pour tout cela, le TDOS offre trois possibilités qui, contrairement aux apparences, ne sont pas identiques, mais plutôt complémentaires. Cependant, disons tout de suite que ces trois possibilités ne concernent pas les fichiers définis par l'instruction ! CREATE (à accès séquentiel ou à accès direct) pour lesquels l'existence de pointeurs empêchent une duplication directe. Nous proposons cependant dans le chapitre concerné un petit programme permettant cette opération.

⊗ La première de ces instructions est :
! COPY « (numéro de disque 1.) nom A (. type 1) = (numéro de disque 2.) nom B (.type 2) »
dont on pourrait résumer la fonction de transfert par :

numéro de disque 1		numéro de disque 2
nom A	émetteur ➧ récepteur	nom B
type 1		type 2

toutes les combinaisons sont possibles entre ces divers paramètres. Cependant (.type 2) peut être absent, auquel cas, par défaut il sera identique à type 1.

Ainsi :
! COPY « 1.FILTRE .BAS = 2.ACTIF .BAS »
permet de dupliquer le fichier (= programme) BASIC de nom FILTRE présent sur une disquette située dans le lecteur n° 1 en un fichier (= programme) BASIC de nom ACTIF sur la disquette située dans le lecteur n° 2.

Cette même instruction pourrait s'écrire :
! COPY « 1.FILTRE .BAS = 2.ACTIF »
ou même :
! COPY « 1.FILTRE = 2.ACTIF »
(.BAS) étant en effet toujours sous entendu par défaut.

Pour les autres cas, toutes les combinaisons de type sont possibles, quoique, par sécurité, il soit préférable que le fichier dupliqué soit du même type que le fichier émetteur. Rappelons au passage que 2 fichiers de même nom mais de types différents sont considérés comme distincts par le TDOS. Une particularité de cette instruction est la suivante : lors de sa réalisation, le fichier concerné est stocké provisoirement en mémoire centrale à la place qu'il occuperait au cours d'une opération ! LOAD (à ce titre, on peut écrire symboliquement :

! COPY « (ndisque 1.)NOMA(.type 1)= (ndisque 2.)NOMB(.type 2) »
équivalent à :
! LOAD « (ndisque 1.)NOMA(.type 1) »
+
! SAVE « (ndisque 2.)NOMB(.type 2) »

La différence avec la réalité étant que, pour l'instruction ! SAVE relative à un fichier binaire, il est nécessaire d'indiquer la zone mémoire concernée).

Ceci présente un inconvénient pour les fichiers binaires stockés en page 4 : leur chargement détruit en effet la partie du TDOS située à cet endroit et bloque le système. Il est donc impossible de recopier un tel fichier avec cette instruction.

⊗ La solution, mais évidemment contraignante, est d'utiliser un utilitaire (attention, ce n'est pas une instruction), appelé « BKP » (en anglais Back-Up) et lancé par !« BKP » situé sur la disquette maître. Son utilisation ne pose pas de problèmes, puisqu'il guide l'utilisateur pas à pas. Il permet de recopier **la totalité** d'une disquette (c'est là la contrainte) soit sur une autre disquette échangée tour à tour avec la précédente dans le

cas d'un seul lecteur, soit sans manipulation, sur une disquette placée dans un second lecteur. L'opération, en effet, ne peut se faire en une seule fois, les informations dupliquées transitant par la mémoire vive de l'unité centrale. Il s'agit-là d'une copie dite : pseudo bit à bit dans la mesure où d'une part, les informations sont copiées indépendamment de leur valeur intrinsèque (« bêtement, mais c'est là un avantage indéniable »), d'autre part seuls les secteurs de la disquette déclarés occupés par les pointeurs adéquats sont copiés.

Il s'agit-là d'ailleurs d'un type de protection possible pour une disquette : lui faire dire qu'elle est vide, alors qu'elle est pleine...

Attention : un back-up détruit la totalité du contenu de la disquette réceptrice. Les fichiers de la disquette émettrice ne sont en effet pas mis à la suite de ceux de la disquette réceptrice, mais à leur place.

⊗ Enfin ! « TKD » permet un dernier type de duplication, un peu particulier, puisqu'il permet de faire passer directement un fichier de cassette à disquette. Il s'agit-là encore d'un utilitaire, dont l'exploitation ne pose pas de problème. Comme pour ! « BKP », l'information transite par la mémoire de l'unité centrale et elle est copiée bit à bit.

Si son utilisation ne pose pas de problèmes pour les fichiers « personnels », tels que des programmes BASIC, elle en pose de nombreux à propos des logiciels du commerce. Ceux-ci en effet atteignent des degrés de sophistication très élevés quant à leur protection. L'une d'entre elle en particulier, est semble-t-il imparable : le programme est chargé en deux morceaux, le premier appelant le second. L'appel se faisant sur cassette, la possession des deux « morceaux » sur disquette ne peut forcément conduire à un résultat. En ajoutant le fait que ces programmes sont enregistrés de plus en plus faiblement (pour éviter la copie sauvage cassette à cassette), on en arrive à des logiciels de 200 F complètement inexploitables, même sur cassette...

Quoiqu'il en soit, l'utilisateur doit être prévenu des déboires auxquels il doit s'attendre lors de ce genre de manipulations.

TDOS ET BASIC

Le TDOS vous permet évidemment, comme le fait un lecteur de cassettes, de charger et de sauvegarder des programmes BASIC, en langage machine en des zones mémoires. Nous allons étudier de plus près ces fonctions élémentaires.

⊗ la commande ! SAVE

Une fois un programme créé, et stocké dans l'unité centrale, il est possible de le sauvegarder sur disquette. L'encombrement minimum sera évidemment d'un secteur sur celle-ci (en fait 2). La commande correspondante contient plusieurs paramètres. Sa syntaxe est :

! SAVE « (numéro de disque.) NOM FICHIER (.TYPE) »

Si vous n'utilisez qu'un lecteur de disquette, le numéro de disque est alors 1 et peut être omis (valeur par défaut). Le NOM FICHIER est le nom que vous désirez donner à votre programme, tel qu'il apparaîtra par la commande ! CAT «» dans le catalogue du disque. Enfin .TYPE n'est pas obligatoire ici, si nous supposons que le fichier à charger est un programme BASIC. Toutefois, .BAS pourrait faire partie de l'instruction.

Ainsi ! SAVE « .FILTRE .BAS » peut être équivalent à ! SAVE « FILTRE ». Comme nous l'avons déjà dit, NOM FICHIER est un texte alphanumérique de 8 caractères maximum, dont le premier est alphabétique.

Cette simple commande est toutefois infiniment supérieure à celle qui lui correspond pour un lecteur de cassettes (CLOAD). En effet, elle peut faire partie intégrante d'un programme, de telle manière qu'un programme en cours puisse en appeller un autre, l'« autre » étant n'importe où sur la disquette, et pas obligatoirement « après », comme cela est nécessaire sur une cassette. Donnons un exemple.

Réalisons les programmes suivants :

⊗ 10 PRINT « je suis le deuxième programme » : B = 10
20 ! LOAD « PROGRAM 3 »
puis ! SAVE « PROGRAM 2 »
et NEW

⊗ 10 PRINT « je suis le premier programme » : A = 5
20 ! LOAD « PROGRAM 2 »
puis ! SAVE « PROGRAM 1 »
et NEW

⊗ Enfin :
10 PRINT « je suis le troisième programme » : C = 15
20 STOP
puis ! SAVE « PROGRAM 3 »
et NEW

⊗ Faisons alors :
! LOAD « PROGRAM 1 »
puis RUN.

« Je suis le premier programme » s'affiche sur l'écran.
Un nouveau RUN conduit à :
« Je suis le second programme» s'affiche.

Un nouveau RUN conduit à :
« Je suis le troisième programme » et un BREAK.
PRINT A,B,C, donne : 0 0 15

Chaque programme en a donc appellé un autre.

Faisons cependant trois remarques :
- Les variables définies dans chaque programme ont été effacées lors de l'appel d'un nouveau programme.
- Chaque nouveau programme a effacé l'ancien
- Il n'est donc pas possible - pour le moment - de concatener 2 programmes.

Nous avons cependant été obligés de faire un RUN après le chargement de chaque nouveau programme. Cela n'aurait-il pu être automatique ?

⊗ la commande ! « (numéro de disque) NOM DE FICHIER (.TYPE) »

Il suffit alors de remplacer les lignes 20 :
20 ! LOAD « PROGRAM X »
par 20 ! « PROGRAM X »
pour que tout se déroule sans intervention exterieure.

La syntaxe est ! « (numéro de disque.) nom du programme (.BAS) »

Cette commande permet donc de lancer un programme basic stocké sur disquette, en l'appelant par le nom sous lequel il est connu dans le catalogue de celle-ci. Les conséquences de ce lancement (pertes des variables,...) sont les mêmes que précédemment ; on peut dire que ! « PROGRAM X » est équivalent à ! LOAD « PROGRAM X » suivi d'un RUN

⊗ la commande ! SAVE «»

C'est l'opération inverse de la précédente et nous l'avons utilisée dans le paragraphe précédent. La totalité des instructions BASIC présente en mémoire centrale est alors transférée sur disquette ; les informations (variables numériques ou de chaîne) sont perdues à ce niveau. Il n'est pas possible de ne charger qu'une partie d'un programme. Rappelons cependant que de nombreux logiciels proposés dans le commerce possèdent une fonction DEL permettant d'éliminer un certain nombre de lignes dans un programme.

La syntaxe de cette commande est :
! SAVE « (numéro de disque.) nom du programme (.BAS) »
Comme les précédentes, elle peut être instruction dans un programme.

Les commandes :
! SAVE « PROGRAM 1 »
! SAVE « 1 .PROGRAM 1 .BAS »
sont correctes.

⊗ la fonction ! RENAME «»

Elle permet de changer le nom d'un programme (ou plus généralement d'un fichier, quel que soit son type) déjà présent sur une disquette. La syntaxe est alors :

! RENAME » (numéro de disque.) ancien nom (.BAS) = nouveau nom (.BAS) »

Ainsi : ! RENAME « PROGRAM 1 = PROG 1 »
et : ! RENAME « 1 .PROGRAM 1 .BAS = PROG 1 .BAS »
sont 2 expressions correctes dans le cas d'un utilisateur ne possédant qu'un seul lecteur de disquette.

Supposons maintenant que l'on veuille mettre au point un programme assez long, dont le nom, par raison de simplicité, sera toujours le même. Il est alors inutile de conserver les différentes versions antérieures, seule la dernière présentant un intérêt. Il y a alors 2 possibilités.

® la commande ! DEL «»
Celle-ci dont la syntaxe est :

! DEL«(numéro de disque.) nom du programme.BAS» permet en effet d'effacer un programme, dans la mesure où celui-ci n'est pas protégé. (Symbole U = UNLOCKED devant le nom du programme dans le catalogue de la disquette). Une fois celle-ci effectuée, le fichier en question a totalement disparu. Attention ! L'indicateur du type, soit : .BAS est ici obligatoire, ceci afin de forcer l'utilisateur à réfléchir par 2 fois avant d'utiliser cette commande.

® la commande ! SAVE «»

En utilisant dans cette commande un nom de fichier déjà utilisé, on **écrase** cet ancien fichier, pour le remplacer par le programme stocké en mémoire centrale. Le même nom peut ainsi supporter les différentes versions d'un programme, mais évidemment seule celle qui correspond à la dernière commande ! SAVE est présente sur la disquette.

Ainsi, la commande :
! SAVE « PROGRAM 1 »

Sous entend toujours :
! DEL « PROGRAM 1 .BAS »
! SAVE « PROGRAM 1 »

Il existe cependant une exception à cette règle : c'est le cas où le fichier présent sur disquette est protégé (symbole L = locked devant le nom du fichier dans le catalogue).

® ! LOCK «» permet donc précisément de verrouiller un fichier, c'est-à-dire d'empêcher toute écriture, effacement ou écrasement d'un fichier (donc d'un programme). Il est bien sûr possible de « lire » un tel fichier protégé.

Il est judicieux à ce propos de verrouiller tous les fichiers essentiels, tels que le TDOS, les utilitaires tels que FORMAT, TKD,...

® ! UNLOCK «» permet l'opération inverse. Par sécurité, il est nécessaire de préciser le type du fichier (soit : .BAS pour un programme Basic) dans la syntaxe de ces 2 dernières fonctions ; celle-ci est la forme classique : ! LOCK « 1 .PROGRAM 1 .BAS ».

⊗ ! COPY «» permet enfin de dupliquer un fichier, c'est-à-dire d'en créer un second identique au premier, sans toutefois effacer celui-ci. Cela permet de créer un nouveau programme, accessible par un nom différent, qui reprend les bases de l'ancien. Cette fonction s'écrit :

! COPY « (numéro de disque.) nom de fichier à copier = nom du nouveau fichier »

(.BAS) est toujours optionnel à la suite des 2 noms.

⊗ la commande ! CUT numéro de ligne

Cette commande permet de supprimer la totalité des lignes d'un programme basic à partir de la ligne numéro de ligne. Grâce à cette commande vous pouvez par l'intermédiaire de l'instruction ! SAVE «» ne sauvegarder qu'une partie de programme.

Mais cette commande a un effet encore plus intéressant : elle ne détruit pas les variables d'un programme en cours d'exécution.

Restriction :

Toutes les variables numériques sont conservées, mais **attention :**

Les seules chaînes alphanumériques qui sont conservées sont celles

- qui sont au-dessus de ! CUT ou
- qui ont été entrées après à partir de INPUT ou
- sur lesquelles des calculs ont été effectués.

Par exemple :

```
10 A$ = « BONJOUR »
20 INPUT « NOM » ; B$
30 B$ = « AU REVOIR »
40 B$ = «» + B$ : B% = 8,
50 C$ = « MERCI » : A = 10,3
60 ! CUT 20
```

La variable C$ est perdue, les autres sont conservées.

Ceci vient du fait que les variables numériques sont stockées dans une zone qui leur est propre, tandis que pour les chaînes alphanumériques l'adresse est directement placée dans le corps du programme. Si la ligne du programme contenant cette adresse a disparu l'ORIC est incapable de retrouver la chaîne. Par contre pour un INPUT et une opération l'ORIC doit stocker les chaînes sur une pile au-dessous de HIMEM ce qui explique que dans ce cas il puisse les retrouver.

Remarque : pour les mêmes raisons que pour les chaînes les DATA sont perdus.

Elle permet donc de libérer de la place en mémoire pour des programmes très longs ou très volumineux.

Dans certains cas ! CUT permet aussi de supprimer de façon dynamique des options dangereuses dans un programme, ou des parties de programme qui ne servent que lors de l'initialisation.

Tapons le programme suivant :

```
10 PRINT "JE SUIS LA PARTIE AVANT !CUT"
20 PRINT
30 PRINT"JE SUIS LA PARTIE APRES !CUT"
40 !CUT 20
```

Tapons RUN une 1re fois : 2 messages s'affichent

```
RUN
JE SUIS LA PARTIE AVANT !CUT

JE SUIS LA PARTIE APRES !CUT
```

RUN une 2e fois : 1 seul message s'affiche

```
JE SUIS LA PARTIE AVANT !CUT
LLIST

10 PRINT "JE SUIS LA PARTIE AVANT !CUT"
```

LIST nous montre que la ligne 10 est la seule restante.

⊗ ! MERGE « nom de programme »

C'est la commande complémentaire de ! CUT, elle permet de concatener 2 programmes BASIC.

Reprenons les 3 petits programmes du chapitre ! SAVE et modifions-les un peu.

On remplace ! LOAD par ! MERGE dans chacun des 3 programmes.

On lance le programme « PROGRAM 1 » comme à l'accoutumé :

```
PROGRAM1

 10 PRINT "JE SUIS LE 1ER PROGRAMME"
 15 A=10.3:C$="QWERTY"
 20 !MERGE"PROGRAM2"
PROGRAM2

 30 PRINT"JE SUIS LE 2IEME PROGRAMME"
 35 B%=5
 40 !MERGE"PROGRAM3"
PROGRAM3

 50 PRINT "JE SUIS LE 3IEME PROGRAMME"
 60 PRINT A,B%,C$
RUN
JE SUIS LE 1ER PROGRAMME
JE SUIS LE 2IEME PROGRAMME
JE SUIS LE 3IEME PROGRAMME
10.3     5        QWERTY
```

Le résultat montre que sans intervention de notre part les 3 programmes se sont déroulés. Mais contrairement au cas où on appelle le programme par ! «» les variables n'ont pas été perdues.

Examinons le contenu de la mémoire programme de l'ORIC par LIST : nous nous apercevons que les trois programmes, sont maintenant présents simultanément ; on a donc bien concatené les trois programmes. On peut sauver le programme ainsi formé comme n'importe quel programme BASIC.

Cette technique permet de faire des programmes à partir d'utilitaires épars et sauvegardés sur disquette. Utilisée conjointement à la commande ! CUT, ! MERGE permet de faire tourner des programmes qui sont plus imposants que la mémoire programme de l'ORIC.

La commande ! MERGE ne peut concatener que des programmes en BASIC donc la mention « .BAS » est facultative.

ATTENTION : le numéro de la 1re ligne d'un programme à merger doit être supérieur au numéro de la dernière ligne déjà en mémoire. Dans le cas contraire le commande ! MERGE fonctionne mais dans votre programme mergé vous aurez plusieurs lignes portant le même numéro ce qui peut être génant en cas de GOTO !!!

- Il existe dans le commerce des utilitaires de renumérotation nous ne saurions trop vous conseiller de vous en procurer un.

APPLICATION DE ! CUT ET ! MERGE

Beaucoup de programmeurs ont l'habitude d'une part, de numéroter leurs logiciels à partir de 10, d'autre part, d'utiliser le même nom générique pour tous les programmes ayant un rapport entre eux. C'est ce que nous avons fait, par exemple, dans le chapitre sur les fichiers à accès séquentiel ; les programmes utilisés s'intitulant PROGN 1, PROGN 2,...

Il est alors intéressant, de pouvoir lister tous ces programmes automatiquement, sans intervention extérieure.

Le principe de l'opération est simple et fait appel à des caractéristiques intéressantes du TDOS. (Voir programme LLIST). Il faut d'abord constituer le nom de chaque programme (par A$), lui associer le programme correspondant (par ! MERGE A$), le lister, l'éliminer (par ! CUT 10) et aller chercher le suivant. Le seul inconvénient est que l'utilisation de LLIST ne permet pas de poursuivre le programme, car cette instruction s'achève toujours par un READY ! Il faut donc inclure un petit programme aimablement mis au point par Monsieur TRAN pour cette occasion en langage machine permettant d'éviter ce phénomène.

Ce programme, relogeable, se compose de 2 parties, l'une placée en # 4500, de 8 octets. Il pourra éventuellement être placé dans le second clavier. # 4500 est appelé juste avant l'instruction LLIST. Le numéro de l'instruction suivant celle-ci est placée en # 36, # 37.

On pourrait également utiliser le principe de ce programme pour obtenir de façon continue le listing de la totalité des programmes d'une disquette. Il faudrait pour cela faire appel au catalogue (voir chapitre sur le langage machine).

#4000	#A9
#4001	#97
#4002	#85
#4003	#35
#4004	#A9
#4005	#30
#4006	#85
#4007	#36
#4008	#A9
#4009	#35
#400A	#85
#400B	#37
#400C	#A9
#400D	#0
#400E	#85
#400F	#38
#4010	#85
#4011	#3A
#4012	#85
#4013	#EA
#4014	#A9
#4015	#B0
#4016	#85
#4017	#1B
#4018	#A9
#4019	#CC
#401A	#85
#401B	#1C
#401C	#A9
#401D	#34
#401E	#85
#401F	#E9
#4020	#4C
#4021	#F4
#4022	#C8

* Ici, il y a Ø
(le numéro de ligne est codé Ø 5)

** Le code ASC II (en hexadécimal)
de 5 (numéro de ligne) est # 35

```
#4500    #A9
#4501    #0
#4502    #85
#4503    #1B
#4504    #A9
#4505    #40
#4506    #85
#4507    #1C
```

*** Ceci est l'adresse du programme précédent

```
1 N=1
2 N$=MID$(STR$(N),2,1):A$="PROGN"+N$:!M
ERGEA$
3 CALL#4500
4 LLIST 10-
5 !CUT10
6 LPRINT:LPRINT A$:LPRINT:LPRINT:LPRINT
:LPRINT
9 N=N+1:GOTO 2
```

** Le code ASC II (en hexadécimal) de 5 (numéro de ligne) est = 35

```
10 !CREATE"SYNDIC,S=1"
20 PRINT:PRINT"Quel est le nombre de co
proprietaires ":INPUT G
30 !WRITE 1'G
40 FOR N = 1 TO G
50 INPUT"Nom du coproprietaire=";A$
60 INPUT"Date du dernier decompte=";B$
70 INPUT"Reliquat a cette date=";SR
80 INPUT"Milliemes=";M
90 INPUT"DU AU =";C$
100 INPUT"SOMME DUE=";SD
110 INPUT"Balance=";B
120 !WRITE 1'N,A$,B$,SR,M,C$,SD,B
130 NEXTN
140 !CLOSE 1
```

PROGN1

```
10 ?OPEN"SYNDIC=1"
20 A$="":B$="":C$=""
25 ?REWIND 1
30 ?TAKE 1'G
40 FOR N=1 TO G
50 ?TAKE 1'N,A$,B$,SR,M,C$,SD,B
60 LPRINT"Nom du coproprietaire.......
............";A$
70 LPRINT"Date du dernier decompte..";B
$
80 LPRINT"Reliquat a cette date.....";S
R;"Fr."
90 LPRINT"Milliemes possedes........";M
100 LPRINT"A la date du ............";
C$
110 LPRINT"Les nouvelles depenses ...";
SD;"FR."
120 LPRINT"conduisent a un SOLDE de....
............";B;"Fr."
130 LPRINT:LPRINT:LPRINT:NEXTN
140 ?CLOSE 1
```

PROGN2

```
10 ?CREATE"DEPENSES,S=1"
20 ?CREATE"FACTURES,S=2"
30 ?CLOSE 0
```

PROGN3

Application du programme précédent

LES FICHIERS DE DONNEES

1° - LES INSTRUCTIONS ! MSAVE et ! MLOAD

Ces 2 instructions, en un sens complémentaires, permettent de stocker et de récupérer les variables d'un programme, dans la mesure où celles-ci font partie d'un tableau (une matrice) ce qui ne présente évidemment aucune difficulté. Elles viennent donc compléter les instructions ! SAVE et ! LOAD, qui elles, ne gèrent que les programmes. Leur syntaxe est :

! MSAVE «(numéro de disque.) nom du fichier = nom de la matrice»

Le nom de fichier est celui que portera la matrice sur la disquette, le nom de la matrice est celui qu'elle a dans le programme. Le DOS donnera alors automatiquement le type .ARY (soit ARRAY en anglais). Ces matrices peuvent être de nature quelconque : tableaux de réels, d'entiers, ou de variables chaînes. La nature cependant doit être précisée dans le nom du tableau. D'autre part, lors de la récupération des informations (par ! MLOAD), il faudra au préalable avoir défini la matrice « réceptrice » par DIM.

OPERATIONS SUR LES MATRICES

Les commandes ! MSAVE et ! MLOAD vous permettent de sauvegarder les matrices, et ceci quel que soit leur type, puis de les récupérer en mémoire centrale. Leur syntaxe est :

! MSAVE « (numéro de disque.) nom du fichier = nom de la matrice » et

! MLOAD « (numéro de disque.) nom du fichier = nom de la matrice ». Le type est automatiquement affecté (.ARY).

Ainsi, les matrices A, B%, E$, définies dans un programme par :
DIMA (5,5)
DIM B% (15)
DIM E$ (8,3,2)
seront sauvegardées respectivement par :
! MSAVE « MA = A »
! MSAVE « MB% = B% »
! MSAVE « ME$ = E$ »

MA, MB%, ME$ sont bien sûr des noms tout à fait arbitraires. Ces noms peuvent éventuellement être les mêmes que celui d'un fichier quelconque, comme un programme BASIC, puisque leur type est alors différent. [(.ARY) et (.BAS)]

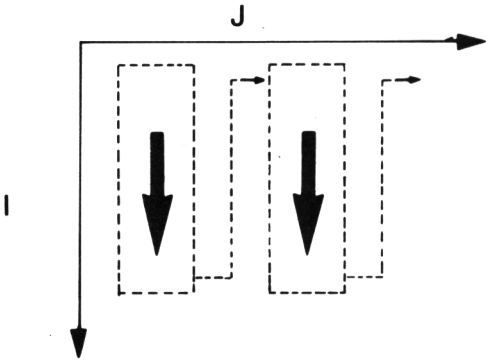
La récupération des éléments d'une matrice stockée sur disquette est quelque peu délicate. Il faut d'abord, bien sûr, que la matrice réceptrice (en mémoire centrale) ait été définie (par DIM) et soit de même type (réel, % ou $).

Ensuite, et sauf subtilité (possible, nous le verrons) de programme, il est préférable d'avoir, pour les 2 matrices en question, les mêmes dimensions.

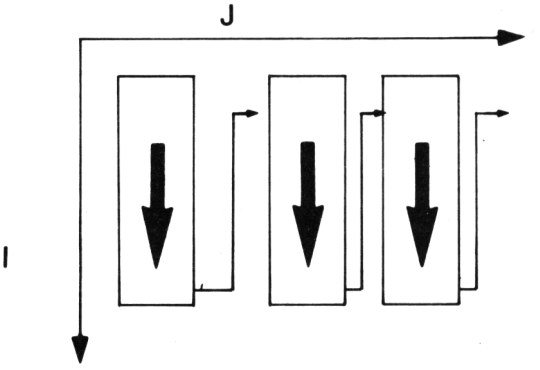
Considérons le programme « MATRICE » dont le listing est donné. Nous définissons une matrice A, dont chaque élément a une valeur distincte des autres. Les lignes 75 à 145 nous permettent de disposer cette matrice sur l'imprimante. Après avoir sauvegardé cette matrice (sous le nom MA), nous effaçons toutes les variables (par CLEAR), récupérons MA et l'affichons de nouveau sur l'imprimante. Tout est correct.

Définissons maintenant une matrice réceptrice C, définie par DIMC (8) (ce qui n'est d'ailleurs pas nécessaire pour une dimension inférieure à 10). Nous constatons que C se remplit dans l'ordre des colonnes de A. De même pour une matrice D [DIM D (4,4)]. Plus curieux est le cas de la matrice E, définie par DIM (11,11).

En conclusion, si les matrices émettrice (sur disquette) et réceptrice (en mémoire centrale) n'ont pas des dimensions identiques, il est possible de prévoir ce qui va se passer :



ordre d'évacuation des données de la matrice émettrice



ordre de remplissage de la matrice réceptrice

MATRICE

```
10 DIM A(10,10)
20 FOR I=0 TO 10
30 FOR J=0 TO 10
40 A(I,J)=N
50 N=N+1
60 NEXT J
```

```
70 NEXT I:GOSUB 75:GOTO150
75 FOR J=0 TO 10:U=9+(5*J)-LEN(STR$(J))
:LPRINT TAB(U) J;:NEXT J:LPRINT:LPRINT
80 FOR I=0 TO 10:LPRINT I;
90 FOR J=0 TO 10
100 L=9+(5*J)-LEN(STR$(A(I,J)))
110 LPRINT TAB(L) A(I,J);
120 NEXT J
130 LPRINT:LPRINT
140 NEXTI
145 RETURN
150 ?MSAVE"MA=A"
160 CLEAR
170 DIM A(10,10)
175 ?MLOAD"MA=A"
180 GOSUB 75
190 CLEAR
200 DIM C(8)
210 ?MLOAD"MA=C"
220 FOR I=0 TO 8
230 LPRINT I,C(I)
240 NEXT I
250 CLEAR
260 DIM D(4,4)
270 ?MLOAD"MA=D"
280 FOR J=0 TO 4:U=9+(5*J)-LEN(STR$(J))
:LPRINT TAB(U) J;:NEXT J:LPRINT:LPRINT
290 FOR I=0 TO 4:LPRINT I;
300 FOR J=0 TO 4
310 L=9+(5*J)-LEN(STR$(D(I,J)))
320 LPRINT TAB(L) D(I,J);
330 NEXT J
340 LPRINT:LPRINT
350 NEXT I
360 CLEAR
```

```
370 DIM E(11,11)
380 ?MLOAD"MA=E"
390 FOR J=0 TO 11:U=9+(5*J)-LEN(STR$(J)
):LPRINT TAB(U) J;:NEXT J:LPRINT:LPRINT
400 FOR I=0 TO 11:LPRINTI;
410 FOR J=0 TO 11
420 L=9+(5*J)-LEN(STR$(E(I,J)))
430 LPRINT TAB(L) E(I,J);
440 NEXT J
450 LPRINT:LPRINT
460 NEXT I
470 END
```

J →

I ↓

	0	1	2	3	4	5	6	7	8	9	10
0	0	1	2	3	4	5	6	7	8	9	10
1	11	12	13	14	15	16	17	18	19	20	21
2	22	23	24	25	26	27	28	29	30	31	32
3	33	34	35	36	37	38	39	40	41	42	43
4	44	45	46	47	48	49	50	51	52	53	54
5	55	56	57	58	59	60	61	62	63	64	65
6	66	67	68	69	70	71	72	73	74	75	76
7	77	78	79	80	81	82	83	84	85	86	87
8	88	89	90	91	92	93	94	95	96	97	98
9	99	100	101	102	103	104	105	106	107	108	109
10	110	111	112	113	114	115	116	117	118	119	120

A (I,J) : Matrice de départ

→ J

↓ I

	0	1	2	3	4	5	6	7	8	9	10
0	0	1	2	3	4	5	6	7	8	9	10
1	11	12	13	14	15	16	17	18	19	20	21
2	22	23	24	25	26	27	28	29	30	31	32
3	33	34	35	36	37	38	39	40	41	42	43
4	44	45	46	47	48	49	50	51	52	53	54
5	55	56	57	58	59	60	61	62	63	64	65
6	66	67	68	69	70	71	72	73	74	75	76
7	77	78	79	80	81	82	83	84	85	86	87
8	88	89	90	91	92	93	94	95	96	97	98
9	99	100	101	102	103	104	105	106	107	108	109
10	110	111	112	113	114	115	116	117	118	119	120

A (I,J) : Matrice stockée sur disquette

J →

I ↓

	0	1	2	3	4
0	0	55	110	45	100
1	11	66	1	56	111
2	22	77	12	67	2
3	33	88	23	78	13
4	44	99	34	89	24

D (I,J)

I ↓

0	0
1	11
2	22
3	33
4	44
5	55
6	66
7	77
8	88

C (I)

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	12	24	36	48	60	72	84	96	108	120	9.47390315E-14
1	11	23	35	47	59	71	83	95	107	119	9.47390315E-14	9.47390315E-14
2	22	34	46	58	70	82	94	106	118	10	9.47390315E-14	9.47390315E-14
3	33	45	57	69	81	93	105	117	9	21	9.47390315E-14	9.47390315E-14
4	44	56	68	80	92	104	116	8	20	32	9.47390315E-14	9.47390315E-14
5	55	67	79	91	103	115	7	19	31	43	9.47390315E-14	9.47390315E-14
6	66	78	90	102	114	6	18	30	42	54	9.47390315E-14	9.47390315E-14
7	77	89	101	113	5	17	29	41	53	65	9.47390315E-14	9.47390315E-14
8	88	100	112	4	16	28	40	52	64	76	9.47390315E-14	9.47390315E-14
9	99	111	3	15	27	39	51	63	75	87	9.47390315E-14	9.47390315E-14
10	110	2	14	26	38	50	62	74	86	98	9.47390315E-14	9.47390315E-14
11	1	13	25	37	49	61	73	85	97	109	9.47390315E-14	9.47390315E-14

E (I,J)

Voici un autre exemple d'opérations sur les matrices :

```
10 REM TESTMAT
11 DIM Q%(5,5)
12 DIM R$(3,3)
13 DIM F(5,5)
15 DIM B%(5,5)
20 DIM A%(5,5)
25 DIM C$(3,3)
30 FOR I=0TO5
40 FOR J=0TO 5
50 A%(I,J)=I+J
55 B%(I,J)=A%(I,J)
60 NEXT J
70 NEXT I
71 FOR I=0 TO 3
72 FOR J=0 TO 3
73 INPUT"C$";C$(I,J)
75 NEXT J:NEXT I
80 !MSAVE"MAR=A%"
90 !MSAVE"MB%=B%"
100 !MSAVE"MC$=C$"
110 !MLOAD"MB%=Q%"
120 !MLOAD"MC$=R$"
130 !MLOAD"MAR=F"
140 FOR I=0 TO 5
150 FOR J=0 TO 5
160 PRINTQ%(I,J);
170 NEXT J:NEXT I
180 PRINT:PRINT"R$"
190 FOR I=0 TO 3
200 FOR J=0 TO 3
210 PRINTR$(I,J)
220 NEXTJ:NEXT I
230 PRINT"F"
240 FOR I=0 TO 5
250 FOR J=0 TO 5
260 PRINTF(I,J);
270 NEXT J:NEXT I
```

ATTENTION :

Dans le cas d'une matrice de chaînes et ceci quelque soit sa dimension, il est nécessaire de définir tous ses éléments avant de faire un ! MSAVE. Par exemple, on pourrait initialiser une matrice A$(3,3) par :

```
- DIM A$(3,3)
- FOR I=0 TO 3
- FOR J=0 TO 3
- LET A$(I,J)=«»
- NEXT J:NEXT I
```

2° - SAUVEGARDES D'ECRANS BASSE RESOLUTION ET HAUTE RESOLUTION ! LSCR«» et ! HSCR«»

Ces deux commandes permettent respectivement de sauver une page vidéo basse résolution ou haute résolution. Ces deux instructions ont besoin d'un seul paramètre le nom sous lequel le fichier sera sauvé. Le type de ces fichiers est SCR pour SCREEN ou écran en anglais. En fait, les informations qui le compose ne sont pas différentes de celles d'un fichier binaire, mais vous êtes dispensés de donner l'adresse de début et de fin de l'écran vidéo que vous voulez sauver.

Par exemple : ! SAVE«IMAGE .BIN,#A000,#B800» et ! HSCR«IMAGE» sont 2 instructions fonctionnellement équivalentes, le premier fichier est du type .BIN dans le catalogue, l'autre est du type .SCR,

Pour rappeler une page déjà mémorisée, il suffit de procéder comme à l'habitude : ! LOAD « nom du fichier .SCR » seul le type est changé : .SCR,

Si l'image ne doit pas être directement chargée dans l'écran où elle a été sauvée : ! LOAD « Nom du fichier .SCR, Adresse de départ ».

Remarques : Avant de charger un écran HIRES il faut se placer dans le mode adéquat. Le passage du mode text au mode HIRES réinitialise tout l'écran haute résolution effaçant les dessins qui s'y trouveraient.

On ne peut pas distinguer un écran texte d'un écran haute résolution autrement que par le nom du programme. Il peut être judicieux d'appeler les écrans « LOW RESOLUTION » L... et les écrans HIGH-Resolution H...

Par exemple LIMAGE et HIMAGE.

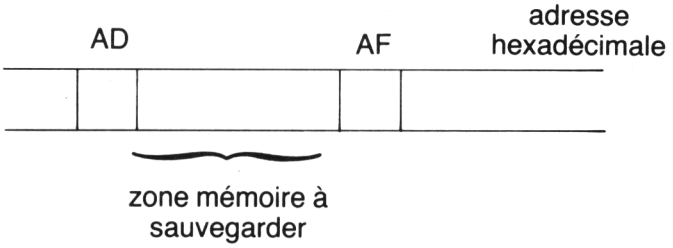
Si vous n'avez pas suivi ce conseil et que vous ne savez plus si un fichier .SCR est basse ou haute résolution sachez que l'écran texte occupe 6 secteurs et l'écran haute résolution 33 secteurs.

Si vous donnez le même nom à un écran texte et à un écran HIRES le dernier enregistrement écrasera le précédant.

3° - AUTRES ZONES MEMOIRES

Indépendamment des cas particuliers précédents, il est évidemment possible de sauvegarder et de récupérer des zones mémoires de l'ORIC, celles-ci pouvant constituer soit un bloc de données binaires, soit un programme en langage machine.

Les instructions ! SAVE et ! LOAD conviennent alors, avec la syntaxe suivante :
! SAVE «(numéro disque.) nom fichier .BIN,AD,AF» pour la première et,
! LOAD «(numéro disque.) nom fichier .BIN,BD» pour la seconde.



Pour l'instruction ! SAVE, il faut préciser les adresses de début (AD) et de fin (AF) de la zone à sauvegarder. Inversement, pour ! LOAD, si le paramètre BD est présent, les données seront chargées à partir de cette adresse. Sinon, les données se retrouveront à la place qu'elles occupaient au moment de la sauvegarde.

Par commodité, .BIN peut être remplacé par .TEXT pour des données binaires, .CMD pour un programme en langage machine.

Dans le cas d'un programme, celui-ci pourra être lancé directement par :
! «(numéro disque.) nom fichier.type(,BD)».

LES INSTRUCTIONS DE TRAITEMENT DE FICHIERS

Il est maintenant classique de distinguer, dans les DOS courants du commerce, deux types de fichiers : d'une part les fichiers à accès séquentiels (S), d'autre part les fichiers à accès direct (D).

1° - QU'EST-CE-QU'UN FICHIER ?

Comme le nom l'indique, cela peut être considéré comme un ensemble de fiches, comportant chacune un certain nombre d'indications. Prenons l'exemple d'un grossiste en quincaillerie désirant gérer son stock sur ordinateur. Il aura bien évidemment un fichier « visserie ». A l'heure actuelle, il s'agit d'un ensemble de fiches en carton, comportant chacune le type de vis concerné, le diamètre, la longueur, et l'état du stock. Veut-il connaître l'état global de celui-ci ? Il lui faut alors consulter chaque fiche, et tuer toutes celles qui font apparaître par exemple, une quantité inférieure à 100. Un client a-t-il besoin de connaître la disponibilité en vis XXY, diamètre 3, longueur 10 ? La fiche concernée le renseignera aussitôt.

Cet exemple donné, il faut bien voir que tout un chacun est concerné par les « fichiers », même si le besoin n'en n'est pas toujours très apparent. Et pourtant ! Le météorologue amateur ? Mais si : la fiche type comprendra la date, la température moyenne du jour, la direction du vent, la pression barométrique, l'hygrométrie... L'enseignant ? Classique : la fiche : nom de l'élève, date de naissance, notes de devoir... L'électronicien ? : trouver rapidement un renseignement sur un composant, d'où la fiche « transistor à effet de champ », qui comportera les indicateurs du genre : telle revue, tel numéro, telle page. Le comptable amateur ? : enregistrer à chaque occasion la date, la nature et le montant des achats effectués.

On pourrait donner ainsi une foule d'exemples. Nous en étudierons quelques uns par la suite dans cet ouvrage. Pour le moment, abandonnons le terme de fiches, pour employer celui d'enregistrements. Un fichier sera donc une série d'enregistrements chacun d'eux pouvant comporter plusieurs paramètres, appelés rubriques, ceux-ci étant de façon générale des variables BASIC tels que des nombres ou des chaînes de caractères.

2° - QUEL TYPE DE FICHIER ?

⊗ critères subjectifs de choix.

Le grossiste en quincaillerie, dont nous parlions tout à l'heure a besoin en général de pouvoir consulter chaque fiche en particulier, tout comme l'enseignant, pour des raisons évidentes. Tous deux ont donc intérêt à créer un **fichier à accès direct,** qui comme le nom l'indique, permet une saisie individuelle de chaque enregistrement. En effet, comme nous le verrons par la suite, chacun d'entre eux porte un « numéro », qui permet donc d'y accéder instantanément. Chaque enregistrement a donc une existence physique réelle sur la disquette et pourrait donc être observé sans ambiguité sur celle-ci avec une très bonne loupe ! La comparaison faite précédemment avec les « fiches » en carton du quincailler est ici extrêmement justifiée. Mais attention : un tel fichier n'a d'intérêt que si toutes les fiches sont sur le même modèle et comportent toutes les mêmes indications, en même nombre.

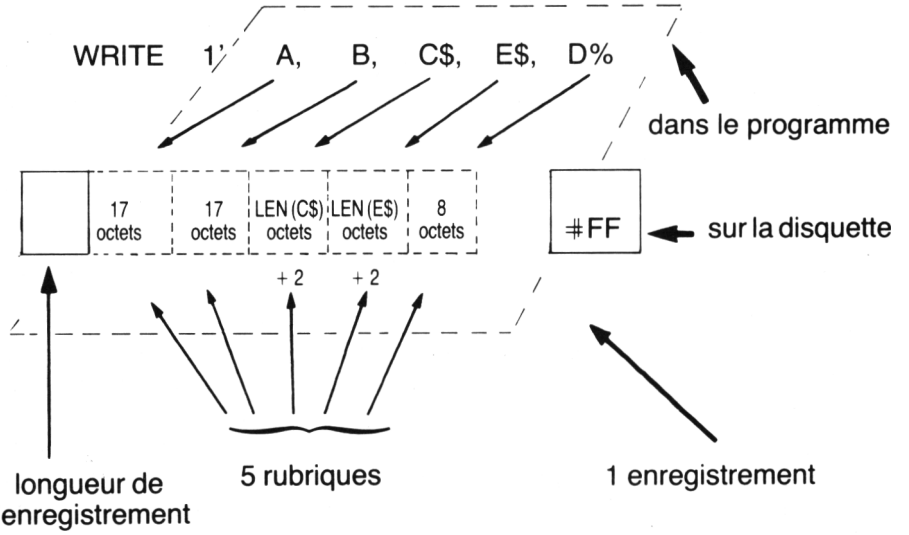
Inversement, le comptable sera plus intéressé d'obtenir à partir de son fichier des balances instantanées (position du compte en banque), des soldes mensuels, annuels... La nature de fiche est ici moins pertinente : il lui suffit de mettre bout à bout, comme il le ferait sur un rouleau de papier : date, nature, montant de chaque opération, l'ensemble de ces 3 données constituant alors un enregistrement, bien que chaque enregistrement puisse aussi comporter plusieurs achats en virements positifs. L'accès direct à un achat particulier n'est ici plus permis (il faut prendre le rouleau de papier « depuis le début »), mais quelle importance ? Seul importe, en effet, le solde instantané, qui nécessite la consultation de la totalité du fichier. Il sera donc judicieux de créer dans ce cas **un fichier à accès séquentiel,** dans lequel tous les enregistrements sont mis bout à bout.

⊗ critères objectifs de choix.

Où se trouve l'intérêt de définir deux types différents de fichier ? Il semblerait, d'après les exemples précédents, que la distinction soit purement théorique. En pratique, il n'en est rien. Peut-on réaliser la tenue d'un

compte en banque à l'aide d'un fichier à accès direct ? Naturellement, et cela permettrait — mais nous l'avons dit l'intérêt n'est pas évident — de connaître les dépenses effectuées tel jour. Mais cela nécessiterait, à la création du fichier :
- de prévoir un enregistrement par jour,
- de prévoir quel nombre de dépenses maximum pourrait être effectué en une journée, ce qui est à priori délicat : elles sont plus nombreuses un jour de soldes que le 20 du mois... Du point de vue du système d'exploitation, ces 2 types de fichiers sont définis de la façon suivante :

⊗ un fichier à accès séquentiel est constitué d'enregistrements distincts, de longueur maximum. 256 octets, eux-mêmes comprenant des rubriques c'est-à-dire en fait des quantités numériques ou alphanumériques. Du point de vue du logiciel, les rubriques sont séparées par une virgule, les enregistrements par un retour à la ligne :



pour les chaînes : 1 caractère = 1 octet
(soit le code ASCII du caractère)

Les enregistrements n'ayant pas tous la même longueur, il n'est pas facile de se positionner avec certitude sur une rubrique donnée. Nous verrons par la suite que cela est possible dans une certaine mesure (instructions ! WHERE, ! REWIND et ! JUMP).

⊗ un fichier à accès direct est constitué d'une suite d'enregistrements, comprenant un certain nombre de rubriques de longueur fixe, eux-mêmes en quantité parfaitement définie à l'avance.

On comprend donc qu'il peut — éventuellement — y avoir un « gachis » effroyable de place si l'on utilise à mauvais escient un fichier à accès direct, pour lequel, en effet, toutes les places non occupées sont de toutes façons réservées sur la disquette. En échange, cette perte de mémoire est compensée par un gain de temps lors de la recherche d'un enregistrement, puisque la place de chacun d'entre eux est parfaitement connu par le logiciel.

MANIPULATION SUR LES FICHIERS A ACCES SEQUENTIEL

1° - INSTRUCTIONS DE TRAITEMENT DYNAMIQUE DES FICHIERS A ACCES SEQUENTIEL

Il faut d'abord créer le fichier. L'instruction est la suivante :

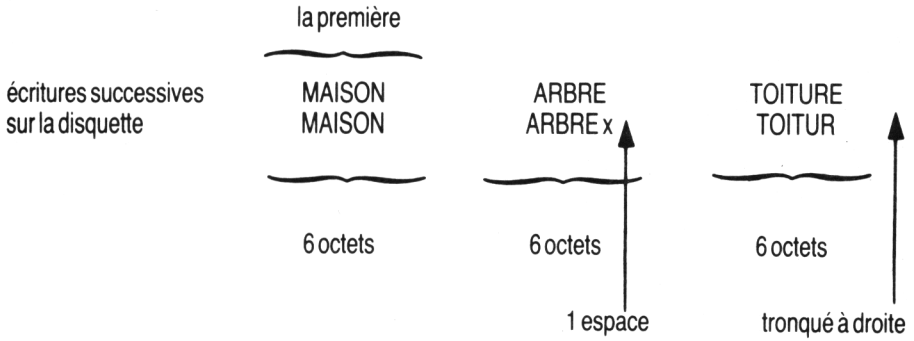
Ⓧ ! CREATE «(numéro de disque.) nom du fichier, S = NLU » NLU est le numéro logique de fichier dont nous avons parlé précédemment. (1 ‹ NLU ‹ 255). Pour le DOS, NLU résume l'ensemble des informations : numéro du disque + nom du fichier + type séquentiel. Cette instruction **ouvre** également le fichier et permet donc d'y accéder, soit en lecture, soit en écriture. Les informations ont automatiquement le type DATA sur la disquette.

Ⓧ ! OPEN « nom du fichier = NLU » permet d'ouvrir un fichier déjà créé. Si cela n'a pas été fait, le message d'erreur « FILE NOT FOUND » vous l'indiquera.

Ⓧ ! WRITE NLU, A,B,C$,E%,D
permet d'écrire à l'enregistrement courant les valeurs des réels A,B, de la chaîne C$, de l'entier E% et du réel D dans **cet ordre.**

Comme nous l'avons vu plus haut, un tel enregistrement, pris comme exemple, a une longueur de :
1 + (17) + (17) + (longueur de C$ + 2) + (8) + (17) + 1 octets.

Il est possible de réécrire sur une rubrique déjà existante. Cependant, il est impératif de ne surcharger une rubrique que par une variable ayant le même type que celle qui était inscrite précédemment. Il faut savoir que, dans ce cas, et contrairement à ce qui se passe dans beaucoup de DOS, une variable chaîne voit sa longueur automatiquement alignée sur celle qui était la première inscrite dans cette même rubrique. Une telle opération ne « bouscule » donc pas le fichier.



⊗ ! TAKE NLU'U,V,Z$,K%,X

permet de saisir (et dans cet ordre) la suite des rubriques indiquées. Il faut bien sûr que le type demandé soit à chaque fois identique à celui de la variable contenue dans la rubrique. D'autre part, un pointeur (dont nous reparlerons à propos de l'instruction ! WHERE) se positionne après chaque lecture sur la rubrique suivante. Ainsi, lire Z$ positionne ce pointeur sur K% et empêche donc en particulier de réécrire Z$.

Enfin, et c'est très important, on ne peut utiliser dans l'instruction ! TAKE que des noms de variable chaîne qui ont été préalablement définis. Ainsi, après une mise sous tension
! OPEN « MAISON = 1 »
! TAKE 1'A$ (si cela est possible)
conduit au message d'erreur :
MISSING VARIABLE
alors que ! OPEN « MAISON = 1 »
A$ = «»
! TAKE 1'A$
est correct

⊗ ! CLOSE 1 ou ! CLOSE 0

permet de fermer un fichier de numéro logique 1 ou bien (seconde expression) de fermer la totalité des fichiers en cours (sachant que l'on ne peut en ouvrir que 16 au maximum). Il n'est alors plus possible de lire ou d'écrire dans un fichier fermé.

⊗ REWIND NLU

permet de se positionner au tout début d'un fichier, de numéro logique NLU, à supposer que celui-ci soit ouvert. En effet, lors de la fermeture d'un fichier, le pointeur se trouve à la rubrique suivant immédiatement la dernière à avoir été lue ou écrite. Cette instruction permet donc, par exemple, de relire un fichier, ou bien de le surcharger.

ATTENTION : Dans une commande ! TAKE ou ! WRITE, il est **essentiel** de ne pas faire suivre un nom de variable par un espace. En effet, ces variables ne pouvant être des variables de tableaux [A$,C%,D autorisés mais A$ (4), C (8) interdits], une telle syntaxe serait considérée par le TDOS comme anormale ; le message d'erreur correspondant serait alors affiché. Ce genre d'erreur étant difficile à appréhender, un espace étant invisible sur l'écran, on pensera donc à ce problème chaque fois qu'un ! TAKE 2'A$ conduit à : SYNTAX ERROR. Le mieux est alors de retaper la commande.

2° - EXEMPLE :

Nous voulons constituer un fichier des fleuves du monde, par ordre décroissant de longueur. Chaque enregistrement comporte 3 rubriques : le nom du fleuve (une chaîne), sa longueur (un entier), son bassin (un réel). Faisons un petit programme pour organiser la saisie des données. Pour le moment, notre fichier ne comprendra que 5 enregistrements.

```
FLEUVES1

 10 !CREATE"FLEUVES,S=2"
 20 FOR N = 1 TO 5
 30 INPUT"Nom du fleuve";A$
 40 INPUT"Longueur.....";L%
 50 INPUT"Bassin.......";B
 60 !WRITE2'A$,L%,B
 70 NEXT N
 80 !CLOSE2
```

Par un RUN, le programme est lancé. Les différentes données sont fournies. Une fois rempli, le fichier est fermé.

Il nous faut maintenant examiner si tout est correct. Nous réalisons un programme « FLEUVES 2 » qui nous permettra d'examiner le contenu du fichier. Ces 2 programmes seront conservés sur disquette (par ! SAVE « FLEUVES 1 » et ! SAVE « FLEUVES 2 »).

```
FLEUVES2

 1 REM "FLEUVES2"
 10 !OPEN"FLEUVES=3"
 20 LPRINTTAB(5)"FLEUVE"TAB(25)"LONGUEUR
TAB(45)"BASSIN"
 30 LPRINT:LPRINT
 40 !REWIND3
 50 FOR N=1 TO 5
 60 C$=""
 70 !TAKE3'C$,S%,J
 80 LPRINTTAB(5)C$TAB(25)S%TAB(45)J
 90 NEXT N
 100 !CLOSE0
```

Ce programme ouvre le fichier, replace le pointeur à son origine (ligne 40) et récupère les informations dans l'ordre. Le résultat est le suivant :

```
FLEUVE                LONGUEUR BASSIN


NIL                    6671     2000000
MISSISSIPPI            6418     3300000
AMAZONE                6280     6120000
YANG-TSE-KIANG         5552     1800000
OB-IRTYCH              5300     3500000
```

Nous nous apercevons alors que nous avons oublié de placer le 6e fleuve de par sa longueur dans le monde, qui, comme chacun le sait, est le Rio de la Plata. Reprenons alors le programme FLEUVES 1, dont nous supprimons les lignes 20 (FOR N = 1 TO 5) et 70 (NEXT N). Un RUN le lance, mais il s'arrête aussitôt sur un message d'erreur :

EXISTING FILE !
BREAK IN 10
Ready

Que s'est-il passé ? En ligne 10, nous constatons qu'il y a l'instruction ! CREATE. Or, le fichier « FLEUVES » a déjà été créé, et ne peut donc l'être de nouveau. Il faut se contenter maintenant de l'ouvrir. (! OPEN). Pour éviter tout problème, nous fermons le fichier (par ! CLOSE 0) et après avoir modifié la ligne 10, nous obtenons le programme suivant :

```
10 !OPEN"FLEUVES=2"
30 INPUT"Nom du fleuve";A$
40 INPUT"Longueur.....";L%
50 INPUT"Bassin.......";B
60 !WRITE2'A$,L%,B
80 !CLOSE2
```

Un examen du fichier par le programme « FLEUVES 2 » dans lequel la ligne 50 a été bien évidemment remplacé par :
FOR N = 1 TO 6
conduit au résultat suivant :

FLEUVE	LONGUEUR	BASSIN
NIL	6671	2000000
MISSISSIPPI	6418	3300000
AMAZONE	6280	6120000
YANG-TSE-KIANG	5552	1800000
OB-IRTYCH	5300	3500000
RIO DE LA PLATA	4700	3100000

La dernière information a bien été inscrite dans le fichier. Remarquons cependant que le dernier enregistrement a été inscrit à la suite des autres ; le fichier avait en effet été fermé après la lecture du cinquième enregistrement.

Il est bien certain que les résulats ainsi obtenus par programme l'auraient été tout aussi bien en direct, instruction par instruction, ce que nous allons faire maintenant, en découvrant 3 nouvelles instructions.

Un fichier ouvert est connu pour le DOS par son numéro logique NLU. Il est maintenant possible de se positionner n'importe où dans un fichier grâce aux instructions : ! REWIND NLU (déjà utilisé), ! APND NLU et ! JUMP NLU, A.

⊗ ! REWIND NLU permet de se placer en début de fichier, sur le premier enregistrement. Il peut alors y avoir lecture (par ! TAKE) ou écriture (avec écrasement) par ! WRITE.

Exécutons la suite des instructions suivantes :

! OPEN « FLEUVES = 1 » [RETURN]
! REWIND 1 [RETURN]
A$=«» [RETURN]
! TAKE 1'A$ [RETURN]
PRINT A$ [RETURN]

ORIC répond : NIL

puis :

! TAKE 1'C% [RETURN]
PRINT C% [RETURN]

conduit à : 6671

On peut ainsi examiner le fichier rubrique par rubrique. Cependant, si la longueur est importante, on peut facilement se perdre. Aussi existe-t-il une instruction très performante permettant de connaître à chaque instant d'une part, la position courante de pointeur de rubrique, et d'autre part le nombre total de rubriques d'un fichier. Tapons maintenant :
! WHERE 1 [RETURN]
ORIC répond :
Ready

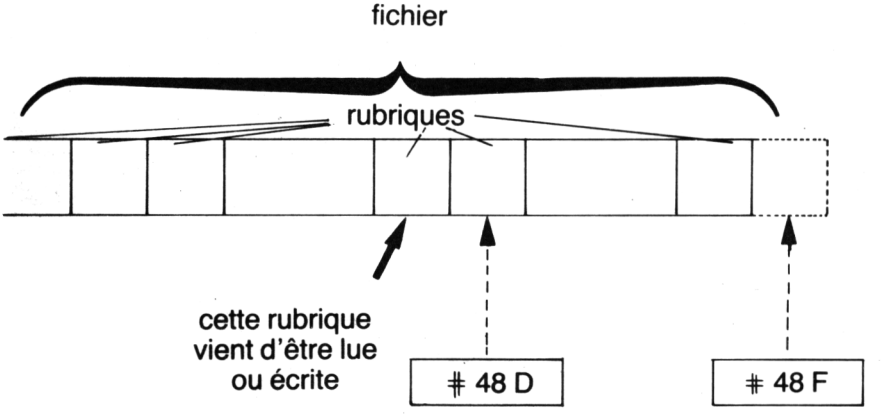
Apparemment, il ne s'est rien passé.

Cependant, les instructions :
PRINT PEEK (# 48 D) : PRINT PEEK (# 48 F)
conduisent à :
3
19
Ready

Le premier nombre indique la position du pointeur de rubrique. En effet, elle est bien de 3 puisque 2 rubriques ont été lues (NIL et 6671) et que la prochaine est la troisième.

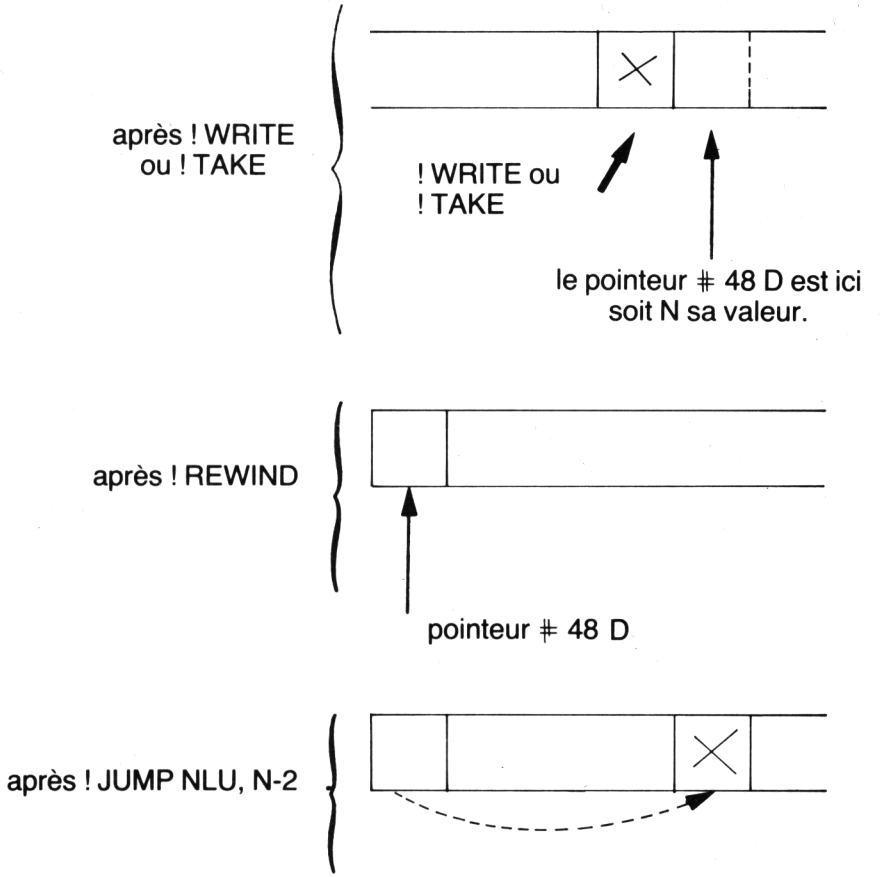
Le second nombre indique le nombre total de rubriques du fichier + 1. Nous avons dans le fichier 6 fleuves et 3 renseignements pour chacun d'entre eux. On a bien : PEEK (# 48 F) = (3 x 6) + 1.

En résumé, à un moment quelconque de l'examen d'un fichier, et après un ! WHERE NLU.



! JUMP NLU, N-2

pour sauter sur une rubrique qui vient d'être lue ou écrite :



⊗ L'instruction ! JUMP NLU, A permet maintenant de se positionner ailleurs dans le fichier, A étant le nombre de rubriques à « sauter ».

Faisons

A = 4 [Return]

! JUMP 1,A [Return]

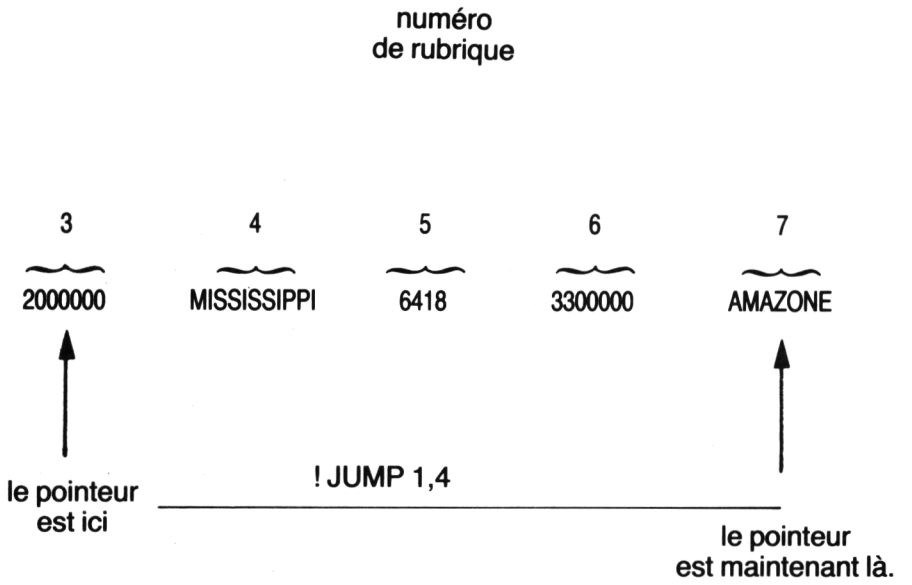
! TAKE 1'A$ [Return]

PRINT A$ [Return]

ORIC répond maintenant :

AMAZONE

En effet



Cette instruction positionne le pointeur à (PEEK # 48D) + A. Remarquons quand même que le numéro de rubrique n'existe nulle part (l'inverse est la propriété des fichiers à accès direct). A priori, seules peuvent être définies la position instantanée du pointeur de rubrique et l'amplitude d'un saut. Rien n'empêcherait cependant l'utilisateur de numéroter ses enregistrements. Nous le verrons par la suite à titre d'exercice.

! APND NLU

Dernière possibilité de positionnement. Cette instruction permet en effet de se placer en fin de fichier, de façon à permetre de rajouter des rubriques. Tapons donc :

! APND1 [Return]

un Ready nous informe que l'opération s'est effectuée. Nous pouvons maintenant réaliser la suite des instructions suivantes :

```
!APND 1
A$="MEKONG"
B%=4500
C=810000
!WRITE 1'A$,B%,C
```

un ! WHERE 1 nous apprend (# 48 D) que le pointeur de rubrique est situé en fin de fichier ; la même quantité apparaît en # 48 F, c'est-à-dire 22. Si nous tentons :
! TAKE 1'A$
le message END OF FILE nous apprend (mais nous le savions) que nous sommes bien en fin de fichier et qu'il n'y a plus rien derrière.

Il est judicieux à ce propos de toujours effectuer un ! APND NLU avant un ! CLOSE NLU de façon à éviter un écrasement involontaire du fichier.

Fermons maintenant ce fichier par ! CLOSE 1 (ou bien ! CLOSE 0).

3° - MANIPULATION SUR LES FICHIERS A ACCES SEQUENTIEL

Nous allons voir maintenant quelques exemples simples d'utilisations des instructions vues précédemment. Nous conserverons le fichier défini dans le chapitre précédent (FLEUVES) pour illustrer ces manipulations. Remarquons deux phénomènes essentiels :

⊗ il est possible de surcharger une rubrique, sachant que le nouveau contenu sera aligné sur la longueur de l'ancien. Le fichier n'est donc pas « bousculé » par une telle opération.

⊗ une rubrique définie comme numérique (réelle ou entière) est en fait occupée par une chaîne, image de ce qui a été tapé au clavier. Ainsi, une rubrique, écrite par ! WRITE NLU'N avec N = 10000 (par exemple) pourra être lue comme une chaîne.
! TAKE NLU'A$
PRINT A$
Conduira à A$ = « 10000 »

Cette remarque importante nous permettra d'examiner un fichier, sans savoir à priori ce que contiennent ses rubriques.

Examen du fichier

a) le fichier est connu.

C'est le cas de « fleuves », dans lequel nous savons qu'il y a 7 enregistrements, composés chacun de 3 rubriques : la première de type chaîne, la seconde de type numérique entier, la troisième de type numérique réel.

Réalisons le programme suivant :

```
FLEUVES3

 1 REM "FLEUVES2"
 10 ?OPEN"FLEUVES=3"
 20 LPRINTTAB(5)"FLEUVE"TAB(25)"LONGUEUR
"TAB(45)"BASSIN"
 30 LPRINT:LPRINT
 40 ?REWIND3
 50 FOR N=1 TO 7
 60 C$=""
 70 ?TAKE3'C$,S%,J
 80 LPRINTTAB(5)C$TAB(25)S%TAB(45)J
 90 NEXT N
 100 ?CLOSE0
```

Il conduit à l'impression suivante :

FLEUVE	LONGUEUR	BASSIN
NIL	6671	2000000
MISSISSIPPI	6418	3300000
AMAZONE	6280	6120000
YANG-TSE-KIANG	5552	1800000
OB-IRTYCH	5300	3500000
RIO DE LA PLATA	4700	3100000
MEKONG	4500	810000

Ce qui est tout à fait normal

b) le fichier est inconnu.

La démarche restera évidemment la même. Mais il nous faudra connaître au départ le nombre de rubriques ; d'autre part, nous ne pourrons pas distinguer les enregistrements.

Supposons que ce soit le cas du fichier « fleuves ». Le programme convenable est alors le suivant :

```
FLEUVES4

1 REM "fleuves4"
10 ?OPEN"FLEUVES=2"
20 ?REWIND2
30 ?WHERE2
40 A=PEEK(#48F)-1
50 LPRINT"Il y a ";A;"rubriques."
60 A$=""
70 FORN=1 TO A
80 ?TAKE2'A$
90 LPRINTA$
100 NEXTN
110 ?CLOSE0
```

un RUN conduit à :

```
Il y a  21 rubriques.
NIL
 6671
 2000000
MISSISSIPPI
 6418
 3300000
AMAZONE
 6280
 6120000
YANG-TSE-KIANG
```

```
 5552
 1800000
OB-IRTYCH
 5300
 3500000
RIO DE LA PLATA
 4700
 31,00000
MEKONG
 4500
 810000
```

Remarquons cependant le décalage (de 1 signe vers la gauche) des grandeurs numériques. L'espace qui apparaît est classique sur ORIC-ATMOS ; il masque le signe + qui a été sous entendu (tous ces nombres sont positifs).

Tentons la suite d'instructions suivantes :

! OPEN « FLEUVES = 1 »
! REWIND 1
! TAKE 1'C$
! TAKE 1'C$
PRINT C$: PRINT VAL (C$)

6671 s'inscrit alors 2 fois sur l'écran. Il est donc ainsi possible de récupérer (par VAL) une variable numérique, dans la mesure où l'on sait que c'en est une.

Terminons par :

! CLOSE 0

4° - RECOPIE DU FICHIER

Nous avons déjà signalé que l'instruction ! COPY ne permet pas de dupliquer un fichier à accès séquentiel ou à accès direct. En effet, à ces types de fichier (type DATA) sont associés des pointeurs qui ne peuvent être transférés simultanément avec les fichiers eux-mêmes. Nous verrons que de toutes façons, ce n'est pas un problème.

Réalisons le programme suivant : (PROGN 6)

PROGN6

```
1 REM "PROGN6"
10 REM RECOPIE DE FICHIER
20 INPUT"fichier de depart";A$
30 INPUT"fichier d'arrivee";B$
40 A$=A$+"=100"
50 !OPENA$
55 !REWIND100
60 B$=B$+",S=101"
70 !CREATEB$
80 C$=""
90 !WHERE100
95 PRINT"Nombre total de rubriques";PEE
K(#48F)-1:PRINT:PRINT:PRINT
96 PRINT"Enregistrement en cours.......
"
100 FOR N=1 TO (PEEK(#48F)-1)
110 !TAKE100'C$
120 PRINTTAB(30)N
125 !WRITE101'C$
130 NEXTN
140 !CLOSE0
150 PRINT:PRINT:PRINT"FIN.............
"
160 STOP
```

Ce programme appelle plusieurs remarques.

® d'abord l'extrême souplesse des instructions du type ! OPEN ou ! CREATE. En effet, la quantité qui les suit peut être définie comme une chaîne A$, ou B$ composées du nom fourni par l'utilisateur et de l'indication du numéro logique de fichier NLU (pris ici égal à 100) pour ! OPEN et de l'indication du type de fichier (S, pour ! CREATE).

⊗ Ensuite, et comme nous l'avons remarqué précédemment, nous transférons toutes les rubriques une par une, sous forme de chaîne, indépendamment de leur nature.

Nous avons donc créé un fichier (qui pourrait s'appeler RIVIERES), identique au fichier « FLEUVES ».

Il serait possible également de créer un fichier « FLEUVES » de même nom, mais sur un lecteur de disquettes autre. Ainsi, le programme suivant permet de générer le paramètre A$ nécessaire à l'instruction ! OPEN, à partir des indications du nom du fichier (fleuves) et du numéro de lecteur (1) fournis par l'utilisateur, ainsi que celui nécessaire à l'instruction ! CREATE, soit B$, à partir du nom souhaité (fleuves) et du numéro de lecteur (2). L'impression de A$ et de B$ confirme tout ceci :

```
A$ ▶        1.FLEUVES=100
B$ ▶        2.FLEUVES2,S=101
```

Un examen (par CAT « 2. ») confirme que ce second fichier a bien été créé.

```
1 REM "PROGN6"
10 REM RECOPIE DE FICHIER
20 INPUT"fichier de depart";A$
25 INPUT"numero de lecteur emetteur";D$
30 INPUT"fichier d'arrivee";B$
35 INPUT"numero de lecteur recepteur";E
$
40 A$=D$+"."+A$+"=100"
50 !OPENA$
55 !REWIND100
60 B$=E$+"."+B$+",S=101"
70 !CREATEB$
80 C$=""
90 !WHERE100
95 PRINT"Nombre total de rubriques";PEE
K(#48F)-1:PRINT:PRINT:PRINT
96 PRINT"Enregistrement en cours.......
"
100 FOR N=1 TO (PEEK(#48F)-1)
110 !TAKE100'C$
```

```
120 PRINTTAB(30)N
125 !WRITE101'C$
130 NEXTN
140 !CLOSE0
150 PRINT:PRINT:PRINT"FIN.............

160 STOP
```

5° - EXTENSION D'UN FICHIER

Il est souhaitable de pouvoir ajouter un enregistrement à un fichier qui a déjà été ouvert. Nous avons déjà vu le cas d'un enregistrement supplémentaire placé en bout de fichier : l'instruction ! APND NLU est bien utile en ce cas. Voyons maintenant comment réaliser cette opération au milieu d'un fichier. Il faut bien sûr savoir à quel endroit doit se placer ce nouvel enregistrement. Pour cela, nous allons ouvrir un fichier tampon, appelé « fleuves 2 » dans lequel nous allons recopier tout le début du fichier « fleuves » ; nous rajouterons ensuite la fiche supplémentaire, puis nous compléterons « fleuves 2 » avec la fin de « fleuves ». « fleuves » sera alors détruit, et « fleuves 2 » transformé en « fleuves ».

```
FLEUVES5

10 !CREATE"FLEUVES2,S=1"
20 !OPEN"FLEUVES=2"
25 !REWIND2
30 INPUT"Nom du fleuve a rajouter";A$
40 INPUT"Sa longueur";S%
50 INPUT"Son bassin";R
60 PRINT:INPUT"Apres quel fleuve voulez
vous le mettre";B$
70 PRINT
80 !WHERE2
90 D=PEEK(#48F)-1
100 E$=""
110 REPEAT
130 !TAKE2'E$,D%,N
```

```
140 ⍰WRITE1'E$,D%,N
150 PRINTTAB(10)E$TAB(20)D%TAB(30)N
160 UNTILE$=B$
170 ⍰WRITE1'A$,S%,R
175 PRINT:PRINTA$,S%,R
180 ⍰WHERE2
190 F=PEEK(#48D)
195 D$=""
200 FORJ=FTOD
210 ⍰TAKE2'D$
220 ⍰WRITE1'D$
230 PRINTD$
240 NEXTJ
250 STOP
```

Le programme « PROGN 7 » nous permet à ce stade d'examiner « FLEUVES 2 ». (Nous avons décidé de rajouter le fleuve « AMOUR » et ses propriétés dans la liste, immédiatement après l'AMAZONE).

PROGN7

```
10 REM EXAMEN D'UN FICHIER
20 INPUT"Nom du fichier.....";A$
30 A$="2."+A$+"=100"
40 ⍰OPENA$
50 ⍰REWIND100
60 ⍰WHERE100
70 FOR N=1 TO (PEEK(#48F)-1)
80 ⍰TAKE100'A$
90 PRINTA$
100 NEXTN
110 ⍰CLOSE100
120 STOP
```

(Remarquez, ligne 30, qu'il est fait appel au second lecteur ; A$ vaut ainsi : « 2.FLEUVES 2 = 100 » ; pour un seul lecteur, il aurait fallu : A$ = « FLEUVES 2 = 100 »

```
NIL
 6671
 2000000
MISSISSIPPI
 6418
 3300000
AMAZONE
 6280
 6120000
AMOUR
 4486
 2050000
YANG-TSE-KIANG
 5552
 1800000
OB-IRTYCH
 5300
 3500000
RIO DE LA PLATA
 4700
 3100000
MEKONG
 4500
 810000
```

Le fleuve AMOUR a bien été rajouté, conformément à nos désirs. Nous pouvons maintenant éliminer le fichier « FLEUVES » pour le remplacer par « FLEUVES 2 ».

Pour cela, faisons d'abord :

! DEL « FLEUVES.DAT »

puis

! RENAME « FLEUVES 2.DAT = FLEUVES.DAT »

(Ne pas oublier le : 2. après les guillemets de chaque instruction si vous faites appel au 2e lecteur). L'opération est maintenant terminée.

6° - SUPPRESSION D'UNE FICHE

Nous allons maintenant faire l'inverse de ce qui a été fait précédemment, c'est-à-dire supprimer un enregistrement. Il nous faudra là également connaître au départ la valeur de l'une quelconque des rubriques de celui-ci. Une première méthode consisterait à utiliser le même procédé que pour l'ajout d'une fiche, c'est-à-dire connaissant la précédente, supprimer celle qui la suit.

Pour varier les plaisirs, nous allons donc procéder autrement. Nous proposons donc de lire les fiches en séquence, de les recopier dans un fichier tampon si elles sont à garder, sauf celle qui est à éliminer. Il faut bien sûr se rappeler que chaque enregistrement contient 3 rubriques. (Nous éliminons l'enregistrement rajouté au paragraphe précédent, c'est-à-dire celui correspondant à AMOUR).

Un dernier examen du fichier (par « PROGN 7 ») nous permet de constater que cette fiche a bien été supprimée.

```
10 REM Suppression d'une fiche
15 INPUT"Quelle fiche voulez vous suppr
imer";D$
20 ?OPEN"FLEUVES=5"
30 ?CREATE"FLEUVES2,S=6'
40 ?REWIND5
50 ?WHERE5
60 N=(PEEK(#48F)-1)/3
65 A$=""
70 FORK=1TON
80 ?TAKE5'A$,G%,J
90 IFA$=D$THEN GOTO110
100 ?WRITE6'A$,G%,J
110 NEXTK
120 ?CLOSE0
130 ?DEL"FLEUVES.DAT"
140 ?RENAME"FLEUVES2.DAT=FLEUVES"
150 ?"PROGN7"
```

```
NIL
 6671
 2000000
MISSISSIPPI
 6418
 3300000
AMAZONE
 6280
 6120000
YANG-TSE-KIANG
 5552
 1800000
OB-IRTYCH
 5300
 3500000
RIO DE LA PLATA
 4700
 3100000
MEKONG
 4500
 810000
```

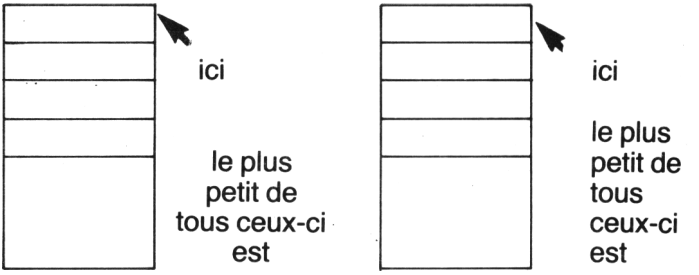
Pour terminer, signalons qu'il est toujours avantageux de recopier (par PROGN 6 par exemple) un fichier avant de le manipuler. Cela permet de conserver un double au cas où un malheur arriverait...

7° - CLASSEMENT D'UN FICHIER

C'est en général le premier travail que l'on désire réaliser sur un fichier. Il faut pour cela connaître le nombre de rubriques par enregistrement et leur type. Un classement alphanumérique ne se fait pas en effet comme un classement numérique !

Reprenons le fichier défini précédemment (fleuves) et tentons de le classer dans l'ordre alphabétique des noms. La méthode dite de « Shell-Metzner », plus rapide, mais mieux adaptée aux fichiers à accès direct, sera vue par la suite.

Le principe de la méthode est le suivant :
partant du premier nom de la liste, nous le comparons à tous les autres. Chaque fois que nous en trouvons un plus « petit » (c'est-à-dire antérieur dans l'ordre alphabétique), nous l'échangeons avec le premier. Il est certain qu'après cette exploration, le plus petit de tous se trouve premier dans la liste. En recommençant pour le 2e, pour le 3e jusqu'à l'avant-dernier, nous classons ainsi la liste.



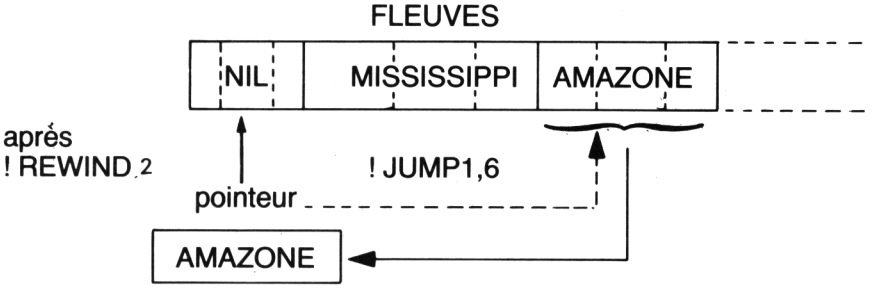
fin de la première passe - fin de la deuxième passe.

Le programme s'explique alors de la manière suivante : on ouvre le fichier « fleuve » non classé évidemment. On recherche alors le nombre N d'enregistrements, en se souvenant que chacun d'entre eux comporte 3 rubriques.

Après avoir dimensionné un certain nombre de tableaux utilisés par la suite, nous transférons les éléments du fichier dans B$ (N), E% (N), F (N). 3 options sont alors proposées : classer le fichier : par ordre alphabétique des noms, par longueur ou par bassin. L'algorithme de tri est évidemment le même dans les 3 cas. Examinons le premier à titre d'exemple. (lignes 300 à 410).

Dans un premier temps, les informations sont stockées dans un tableau « tampon » K$ (N). Un tableau E (N) d'autre part, permet de se souvenir de l'ordre initial. Puis, pour chaque valeur d'un indice I (1 ‹ I ‹ N-1), chaque élément correspondant est comparé à tous les suivants, et interchangé avec eux suivant le résultat de leur comparaison. Le tri terminé, un fichier tampon « FLEUVES 2 » est créé. E (N) contenant maintenant l'ordre du classement, il faut aller chercher successivement chaque enregistrement de « FLEUVES » dans cet ordre pour remplir « FLEUVES 2 ». Les lignes 990 à 1020 permettent maintenant d'examiner « FLEUVES 2 » et de s'assurer que le tri a été fait.

Exemple E (1) = 3 ➧ AMAZONE



Rappelons que les ! ERSET et ! EROFF des lignes 935 et 945 évitent un blocage du programme lorsque Y = O conduit à un ! JUMP interdit. Le programme est donné à la suite, ainsi que le résultat du déroulement du programme pour les 3 options possibles. Enfin, signalons que lors de la mise au point d'une manipulation sur un fichier, il est prudent — et fortement recommandé ! — de dupliquer au départ le fichier en question. En cas de fausse manœuvre, il en restera ainsi une trace...

```
FLEUVES6

 10 !OPEN"FLEUVES=2"
 20 !WHERE2
 30 A$=""
 40 N=(PEEK(#48F)-1)/3
 45 PRINTN
 50 DIMB$(N):DIME%(N):DIMF(N):DIME(N):DI
MK$(N):DIMH%(N):DIMG(N)
 60 !REWIND2
 70 FOR L=1 TO N
 80 !TAKE2'A$,M%,U
 90 B$(L)=A$
 100 E%(L)=M%
 110 F(L)=U
 120 NEXTL
 130 PRINT"Voulez vous classer ce fichie
r en fonction:"
```

```
 140 PRINT"          du nom des fleuves (
1)"
 150 PRINT"           "
 160 PRINT"          de leur longueur (2)
"
 170 PRINT"           "
 180 PRINT"          de leur bassin (3) ?
"
 190 PRINT:GETA
 200 ON A GOSUB 300,500,700:GOTO900
 300 REM Classement par nom
 310 FOR L=1TON
 320 K$(L)=B$(L):E(L)=L
 330 NEXTL
 340 FORI=1TON-1
 350 FORJ=I+1 TO N
 360 IF (K$(J) < K$(I)) THEN 380
 370 NEXTJ:NEXTI
 375 RETURN
 380 K1$=K$(I):K2=E(I)
 390 K$(I)=K$(J):E(I)=E(J)
 400 K$(J)=K1$:E(J)=K2
 410 GOTO370
 500 REM Classement par longueur
 510 FOR L=1TON
 520 H%(L)=E%(L):E(L)=L
 530 NEXTL
 540 FORI=1TON-1
 550 FORJ=I+1 TO N
 560 IF (H%(J) < H%(I)) THEN 580
 570 NEXTJ:NEXTI
 575 RETURN
 580 E1=H%(I):K2=E(I)
 590 H%(I)=H%(J):E(I)=E(J)
 600 H%(J)=E1:E(J)=K2
```

```
610 GOTO570
700 REM Classement par bassin
710 FOR L=1TON
720 G(L)=F(L):E(L)=L
730 NEXTL
740 FORI=1TON-1
750 FORJ=I+1 TO N
760 IF (G(J) < G(I)) THEN 780
770 NEXTJ:NEXTI
775 RETURN
780 E1=G(I):K2=E(I)
790 G(I)=G(J):E(I)=E(J)
800 G(J)=E1:E(J)=K2
810 GOTO770
900 ?CREATE"FLEUVES2,S=3"
910 FOR L=1TON
920 ?REWIND2
930 Y=E(L)*3:Y=Y-3
935 ?ERSET
940 ?JUMP2,Y
945 ?EROFF
946 C$=""
950 ?TAKE2'C$,M%,U
955 PRINTC$:LPRINTC$
960 ? WRITE3'C$,M%,U
970 NEXTL
975 LPRINT:LPRINT
980 ?REWIND3
990 FORL=1TON
1000 ?TAKE3'A$,B%,C:PRINTA$
1010 LPRINTTAB(1)A$TAB(20)B%TAB(40)C
1020 NEXTL
1030 ?CLOSE0
```

AMAZONE
MEKONG
MISSISSIPPI
NIL
OB-IRTYCH
RIO DE LA PLATA
YANG-TSE-KIANG

AMAZONE	6280	6120000
MEKONG	4500	810000
MISSISSIPPI	6418	3300000
NIL	6671	2000000
OB-IRTYCH	5300	3500000
RIO DE LA PLATA	4700	3100000
YANG-TSE-KIANG	5552	1800000

MEKONG
RIO DE LA PLATA
OB-IRTYCH
YANG-TSE-KIANG
AMAZONE
MISSISSIPPI
NIL

MEKONG	4500	810000
RIO DE LA PLATA	4700	3100000
OB-IRTYCH	5300	3500000
YANG-TSE-KIANG	5552	1800000
AMAZONE	6280	6120000
MISSISSIPPI	6418	3300000
NIL	6671	2000000

```
MEKONG
YANG-TSE-KIANG
NIL
RIO DE LA PLATA
MISSISSIPPI
OB-IRTYCH
AMAZONE
```

MEKONG	4500	810000
YANG-TSE-KIANG	5552	1800000
NIL	6671	2000000
RIO DE LA PLATA	4700	3100000
MISSISSIPPI	6418	3300000
OB-IRTYCH	5300	3500000
AMAZONE	6280	6120000

8° - TRAITEMENT D'ERREURS

Le TDOS dispose d'un puissant système de traitement d'erreurs, mais seulement pour ses propres instructions (celles commençant par un : !) On sait que cette fonction intéressante ne fait pas partie du BASIC de l'ORIC.

Qu'est-ce-qu'une erreur ? Il s'agit d'un blocage du fonctionnement du TDOS, car il a rencontré une instruction qu'il ne comprend pas, qu'il ne sait pas exécuter ou bien qu'il ne peut pas exécuter. Considérons l'instruction suivante :
! SAVA « FLEUVES »

Il y a évidemment une erreur de syntaxe dans cette commande. Introduite directement au clavier, et après un RETURN, s'affiche un message dit d'« erreur » :
SYNTAX ERROR
BREAK
Ready

Dans un programme, il y aurait eu rupture de séquence et le programme se serait arrêté automatiquement (sous contrôle du DOS) sur cette instruction. Ce pourrait être un inconvénient.

Considérons un autre exemple.

Lors de l'exploration des fichiers, dans les paragraphes précédents, nous avons fait remarquer qu'il était possible de connaître à priori le nombre de rubriques qu'ils contenaient (par ! WHERE NLU). Cela permet de savoir à chaque instant combien de rubriques il reste avant d'aboutir à la fin du fichier.

Cependant, s'il y a une erreur involontaire dans le décompte, une tentative de lecture à la première rubrique non enregistrée se traduira par un message d'erreur « END OF FILE » qui bloquera le programme en ce point. C'est forcément ce qui se produira avec le petit programme suivant :

```
10 ! OPEN « FLEUVES » = 100 »
20 ! REWIND 100
30 ! TAKE A$
40 ! GOTO 30
```

Au bout d'un certain temps, le programme s'arrête par :

```
END OF FILE
BREAK IN 30
Ready
```

Cependant, il est visible que dans ce cas, l'« erreur » n'est pas grave — elle est même normale ! — et ne constitue pas un blocage du système. Il est donc possible avec le TDOS de contrôler ce genre d'erreurs, de manière à pouvoir poursuivre l'exécution d'un programme.

La première instruction relative à ce traitemennt d'erreur est : ! ERSET. Placée dans un programme en un point quelconque, elle empêche à partir de ce point l'affichage des messages d'erreur et les BREAK (arrêts de programme) qui en découlent. Les instructions ayant éventuellement généré une « erreur » sont purement et simplement sautées et les programmes se poursuivent par les instructions suivantes. L'instruction en cause n'est donc **pas exécutée.**

La fonction inverse est ! EROFF. A partir de l'instruction qui la contient, toute erreur entraîne l'émission du message correspondant et un arrêt du programme.

Cependant, utilisée sans discernement, l'utilisation de telles commandes peut être désastreuse suivant l'erreur qui s'est produite. Un I/O ERROR ou un DISK FULL ER n'ont pas les mêmes conséquences

qu'un END OF FILE bien maîtrisé ! Aussi, est-il conseillé de toujours utiliser dans la partie de programme occultée par ! ERSET l'instruction : ! ERR GOTO N qui elle va nous permettre de vérifier le type de l'erreur. En effet, l'utilisation de cette instruction génère l'apparition d'un code d'erreur dans la mémoire # 489. (voir annexe : les messages d'erreur et leur code, en fin de livre). L'examen de ce code d'erreur permettra de décider ou nom de la poursuite du programme.

Considérons le petit exemple suivant :

```
FLEUVES?

 10 !ERSET
 20 A$=""
 30 !OPEN"FLEUVES=100"
 40 !TAKE[00]A$.PRINTA$
 50 !ERR GOTO70
 60 GOTO40
 70 ER=PEEK(#489)
 80 IF ER=15 THEN GOTO90 ELSE PRINT"ERRE
UR GRAVE":!CLOSE0:STOP
 90 PRINT"Tout s'est bien passe."
 100 !CLOSE0:STOP
```

Il permet d'examiner le contenu d'un fichier, sans se préoccuper du nombre de rubrique qu'il contient. La seule erreur, prévisible et justifiée, est celle de « fin de fichier » ; il faut donc le programmer pour que tout se déroule normalement.

Un RUN nous permet effectivement de vérifier le contenu du fichier « FLEUVES ». A la fin, s'affiche le message « tout s'est bien passé » car au moment où l'on a tenté de lire la première rubrique non enregistrée (ligne 40), opération qui ne s'est naturellement pas exécutée, le traitement d'erreur (ligne 80) a cherché à vérifier si l'erreur trouvée est bien celle qui était attendue (code 15 : END OF FILE). Dans ce cas les lignes 90 et 100 ont été normalement exécutées. Si l'erreur avait été d'une autre nature, chose à priori impossible ici, il aurait fallu arrêter le traitement dès son apparition pour éviter un événement aux conséquences imprévisibles.

Nous pouvons tenter d'améliorer le programme précédent en utilisant l'instruction ! SEARCH.

⊗ ! SEARCH « (numéro de disque.) **nom du fichier.type** ». Cette instruction a ceci de particulier que la seule conséquence de son utilisation est dans tous les cas l'apparition d'un message d'erreur (et donc l'arrêt du programme en cours, sauf si ! ERSET a été préalablement employée).

En mode commande, le message « EXISTING FILE » s'affichera si le fichier existe, avec son type (donc tel qu'il apparaît dans le catalogue par ! CAT «») ; sinon c'est le message « FILE NOT FOUND » qui apparaîtra. Les conséquences sont identiques en mode programme, avec arrêt du programme à la ligne contenant cette instruction.

Si ! ERSET a empêché ce type de réaction, la suite du programme doit tout de même prévoir la conduite à tenir suivant que le fichier en question existe ou non !

Le programme suivant, amélioration du précédent, le démontre :

```
FLEUVES8

 10 !ERSET
 12 A$=""
 14 INPUT"NOM DU FICHIER";B$
 16 B$=B$+".DAT":!SEARCHB$
 18 !ERR GOTO70
 20 !OPEN"FLEUVES=100"
 30 !REWIND100
 40 !TAKE100'A$:PRINTA$
 50 !ERR GOTO78
 60 GOTO40
 70 ER=PEEK(#489)
 75 IF ER = 1 THEN GOTO20 ELSE PRINT"FIC
HIER INEXISTANT":STOP
 78 ER=PEEK(#489)
 80 IF ER=15 THEN GOTO90 ELSE PRINT"ERRE
UR GRAVE":!CLOSE0:STOP
 90 PRINT"Tout s'est bien passe."
 100 !CLOSE0:STOP
```

Il y a d'abord interrogation de l'utilisateur sur le fichier qu'il veut examiner. Si ce fichier existe (EXISTING FILE, Code 1), il est examiné comme dans l'exemple précédent. S'il n'existe pas, la recherche est arrêtée (ligne 75). Le STOP de cette même ligne pourrait d'ailleurs être remplacé par un GOTO 14, permettant ainsi à l'utilisateur de proposer un nouveau nom de fichier.

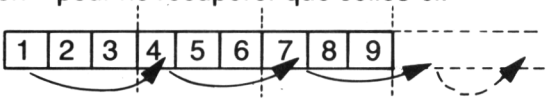
Remarquons cependant qu'une erreur de syntaxe ligne 16 (le point entre B$ et DAT) génèrera systématiquement une SYNTAX ERROR (code 11) ; le programme bouclera alors indéfiniment sur le message « FICHIER INEXISTANT », même si le catalogue indique sa présence !

Il est donc essentiel de contrôler toutes les erreurs susceptibles de se présenter avant d'utiliser l'instruction ! ERSET. Une erreur de syntaxe dans un programme est impardonnable à ce niveau ; il peut être utile de faire « tourner » un programme, sans ! ERSET quelquefois afin de déterminer toutes les erreurs imprévisibles susceptibles de se produire.

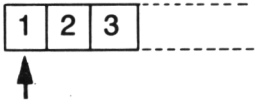
A ce sujet, il peut être intéressant, lorsque de nombreux codes d'erreur sont examinés dans un programme, d'utiliser la commande TRON de l'ATMOS, qui permet d'examiner le « cheminement » mental de l'ensemble des opérations qui sont effectuées.

Montrons sur un exemple différent l'intérêt de ces commandes. Reprenons le fichier « fleuves » et cherchons à en extraire tous les noms, et seulement eux. Deux solutions s'offrent à nous :

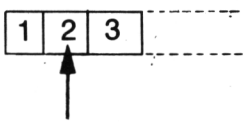
- ⊗ Ouvrir le fichier, replacer les pointeurs en son début et, sachant que chaque enregistrement est composé de 3 rubriques, dont à chaque fois, seule la première nous intéresse, faire des « sauts de mouton » pour ne récupérer que celles-ci.



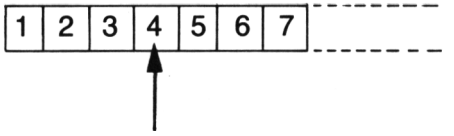
On remarquera qu'après un TAKE le pointeur (situé en # 48 D) est automatiquement incrementé, ce qui fait que, contrairement aux apparences, les JUMP doivent être de 2 unités, et non de 3.



après le REWIND



après le TAKE



après le JUMP

Le programme s'en déduit aisément :

```
FLEUVES9

1000 !OPEN"FLEUVES=2"
1010 !REWIND2
1020 !WHERE2
1030 S=PEEK(#48F)-1
1040 FORJ=0TOS-1STEP3
1050 A$=""
1052 !TAKE2'A$
1055 !JUMP2,2
1065 !WHERE2:PRINTPEEK(#48D)
1070 PRINTA$
1080 NEXTJ
1090 !CLOSE2
```

et ses conséquences :

Valeur de J — Conséquences des TAKE

```
0   NIL
3   MISSISSIPPI
6   AMAZONE
9   YANG-TSE-KIANG
12  OB-IRTYCH
15  RIO DE LA PLATA
18  MEKONG
```

⊗ Une autre méthode consisterait à « rebobiner » le fichier après chaque extraction de rubrique, en se positionnant successivement sur la 1re, 4e, 7e rubrique. Cela correspondrait, après un REWIND à des JUMP respectifs de 0, 3, 6,... Or, O est une valeur interdite pour un JUMP. Une telle tentative conduit immanquablement à « RANGE ERROR », puis un BREAK.

Ayant donc localisé une source d'erreur « naturelle », il nous est possible de la masquer ; il faudra donc l'encadrer par ! ERSET et ! EROFF. D'où le programme, et ses conséquences :

```
FLEUVE10

1000 !OPEN"FLEUVES=2"
1010 !REWIND2
1020 !WHERE2
1030 S=PEEK(#48F)-1
1040 FORJ=0TOS-1STEP3
1050 A$=""
1052 !REWIND2
1054 !ERSET
1055 !JUMP2,J
1060 !TAKE2'A$
1065 !WHERE2:PRINTPEEK(#48D)
```

```
1070 PRINTA$:LPRINTJ,A$
1080 NEXTJ
1090 ?CLOSE2
```

Valeur de départ de # 48 D après un REWIND : 1

Valeur du saut	Rubrique concernée
0	NIL
3	MISSISSIPPI
6	AMAZONE
9	YANG-TSE-KIANG
12	OB-IRTYCH
15	RIO DE LA PLATA
18	MEKONG

8° - PREMIERE APPLICATION : utilisation didactique du fichier « fleuves ». Nous nous proposons d'utiliser le fichier « fleuves » pour réaliser un questionnaire (simple) de géographie : l'ordinateur fournissant aléatoirement un nom de fleuve (chacun une fois et une seule), il demande au questionné la longueur du fleuve et détermine si la réponse est juste ou pas. Le programme comprend plusieurs parties :

⊗ Lignes 20 à 90 : sous programme de choix d'un nombre aléatoire compris 1 et 7. Ce sera pour la suite le numéro de l'enregistrement concerné.

⊗ Lignes 100 à 220 : présentation du problème et des 7 fleuves qui seront utilisés dans le questionnaire.

⊗ Lignes 300 à 340 : l'utilisateur propose une valeur possible pour la longueur de ce fleuve ; l'ordinateur décide de la validité de la proposition.

FLEUVE11

```
20 U=(INT(RND(1)*7+1)):V(J)=U
25 Z=0
26 IF J=1THEN70
30 FORL=1 TOJ-1
40 IF V(L)=UTHEN Z=1
50 NEXTL
60 IFZ=1THEN GOTO20
70 PRINTU
90 RETURN
99 REM ****************************
100 REM Programme
101 REM****************************
110 CLS:PRINT:PRINT:PRINT
120 PRINT"JE VAIS VOUS PROPOSER UN FLEU
VE CHOISI DANS LA LISTE SUIVANTE:"
130 PRINT:PRINT:PRINT
140 ?OPEN"FLEUVES=2"
150 ?REWIND2
160 ?WHERE2
170 S=PEEK(#48F)-1
175 A$=""
180 FORJ=0TOS-1STEP3
190 ?TAKE2'A$
200 ?JUMP2,2
210 PRINTA$
220 NEXTJ
240 FORJ=1TO7
250 GOSUB20
260 ?REWIND2
270 V=(3*U)-3
275 ?ERSET
280 ?JUMP2,V
285 ?EROFF
```

```
290 ?TAKE2'A$,D%
300 PRINT:PRINT:PRINT:PRINT"Quelle est
la longueur du fleuve ";A$;" ?"
310 INPUT K%
320 IF K% < 1.05*D% AND K% > 0.95*D% TH
ENPRINT"BONNE REPONSE":GOTO 350
330 PRINT"MAUVAISE REPONSE"
340 PRINT"IL FALLAIT REPONDRE:";D%
350 NEXTJ
```

```
JE VAIS VOUS PROPOSER UN FLEUVE CHOISI D
ANS LA LISTE SUIVANTE:

NIL
MISSISSIPPI
AMAZONE
YANG-TSE-KIANG
OB-IRTYCH
RIO DE LA PLATA
MEKONG


Quelle est la longueur du fleuve OB-IRTY
CH ?
 5250
BONNE REPONSE


Quelle est la longueur du fleuve RIO DE
LA PLATA ?
 4600
BONNE REPONSE
```

```
Quelle est la longueur du fleuve MEKONG
?
 4500
BONNE REPONSE


Quelle est la longueur du fleuve AMAZONE
 ?
 6280
BONNE REPONSE


Quelle est la longueur du fleuve YANG-TS
E-KIANG ?
 5500
BONNE REPONSE


Quelle est la longueur du fleuve NIL ?
 2000
MAUVAISE REPONSE
IL FALLAIT REPONDRE: 6671


Quelle est la longueur du fleuve MISSISS
IPPI ?
 6000
MAUVAISE REPONSE
IL FALLAIT REPONDRE: 6418
```

EXEMPLE D'UTILISATION DES FICHIERS A ACCES SEQUENTIEL

Nous allons examiner la manière dont on pourrait utiliser les fichiers séquentiels, pour gérer les comptes d'un syndic de copropriété. Pourquoi ne pas utiliser à accès direct ? Il y a plusieurs raisons à cela.

- Pour un syndic, tous les copropriétaires sont équivalents. Il n'a pas besoin en général d'avoir accès à un compte particulier. Seul l'intéresse en fin de semestre un examen séquentiel de son fichier pour connaître le montant des appels de fonds qu'il doit émettre.

- De même, une dépense particulière ne joue pas un grand rôle individuel. Il y a en effet d'une part, l'ensemble des factures à payer et d'autre part, l'ensemble des dépenses couvertes. Leur connaissance globale permet de faire des bilans et de fournir des justificatifs.

- La lenteur d'exécution n'est pas un problème ici, dans la mesure où il faut de toutes façons réaliser un examen séquentiel des fichiers à intervalles réguliers.

Il semblerait donc nécessaire de créer 3 fichiers :

⊗ un fichier compte client, contenant pour chaque enregistrement, le nom, le numéro d'appartement correspondant, la date de l'ancien décompte, l'imputation de dépenses en fonction des millièmes possédés par le copropriétaire, la balance mise à jour.

Le programme correspondant pourrait être le suivant :

```
PROGN1

 10 ?CREATE"SYNDIC,S=1"
 20 PRINT:PRINT"Quel est le nombre de co
proprietaires ":INPUT G
 30 ?WRITE 1'G
 40 FOR N = 1 TO G
 50 INPUT"Nom du coproprietaire=";A$
 60 INPUT"Date du dernier decompte=";B$
 70 INPUT"Reliquat a cette date=";SR
 80 INPUT"Milliemes=";M
 90 INPUT"DU AU =";C$
 100 INPUT"SOMME DUE=";SD
 110 INPUT"Balance=";B
 120 ?WRITE 1'N,A$,B$,SR,M,C$,SD,B
 130 NEXTN
 140 ?CLOSE 1
```

Après la création de ce fichier, il serait utile de pouvoir le consulter. C'est l'objet du programme suivant.

```
PROGN2

 10 ?OPEN"SYNDIC=1"
 20 A$="":B$="":C$=""
 25 ?REWIND 1
 30 ?TAKE 1'G
 40 FOR N=1 TO G
 50 ?TAKE 1'N,A$,B$,SR,M,C$,SD,B
 60 LPRINT"Nom du coproprietaire.......
..............";A$
```

```
 70 LPRINT"Date du dernier decompte..";B
$
 80 LPRINT"Reliquat a cette date.....";S
R;"Fr."
 90 LPRINT"Milliemes possedes........";M
 100 LPRINT"A la date du .............";
C$
 110 LPRINT"Les nouvelles depenses ...";
SD;"FR."
 120 LPRINT"conduisent a un SOLDE de....
.............";B;"Fr."
 130 LPRINT:LPRINT:LPRINT:NEXTN
 140 ?CLOSE 1
```

Il est certain qu'après la création, un certain nombre de rubriques ne sont pas remplies. Elles le seront par la suite, lors de la mise à jour semestrielles des dépenses.

Exécution de PROGN 2

```
Nom du coproprietaire........DUPONT
Date du dernier decompte..01/01/84
Reliquat a cette date.....-500 Fr.
Milliemes possedes........ 125
A la date du .............01/01/84
Les nouvelles depenses ... 0 FR.
conduisent a un SOLDE de........-500 Fr.

Nom du coproprietaire........DURAND
Date du dernier decompte..01/01/84
Reliquat a cette date..... 1000 Fr.
Milliemes possedes........ 180
A la date du .............01/01/84
Les nouvelles depenses ... 0 FR.
conduisent a un SOLDE de........ 1000 Fr
```

```
Nom du coproprietaire........BRUN
Date du dernier decompte..01/01/84
Reliquat a cette date..... 300 Fr.
Milliemes possedes........ 295
A la date du .............01/01/84
Les nouvelles depenses ... 0 FR.
conduisent a un SOLDE de........ 300 Fr.
```

```
Nom du coproprietaire........BELLO
Date du dernier decompte..01/01/84
Reliquat a cette date.....-800 Fr.
Milliemes possedes........ 250
A la date du .............01/01/84
Les nouvelles depenses ... 0 FR.
conduisent a un SOLDE de........-800 Fr.
```

```
Nom du coproprietaire........DRENO
Date du dernier decompte..01/01/84
Reliquat a cette date..... 700 Fr.
Milliemes possedes........ 150
A la date du .............01/01/84
Les nouvelles depenses ... 0 FR.
conduisent a un SOLDE de........ 700 Fr.
```

⊗ Deux fichiers distincts, l'un pour les dépenses à couvrir, l'autre pour les factures (archives).

Il faut d'abord les créer.

```
PROGN3

 10 ºCREATE"DEPENSES,S=1"
 20 ºCREATE"FACTURES,S=2"
 30 ºCLOSE 0
```

Puis, les mettre à jour :

```
PROGN4

 10 ºOPEN"DEPENSES=1"
 15 ºOPEN"FACTURES=2"
 20 ºAPND 0
 30 PRINT"Voulez vous ajouter une depens
e (O/N)?":GET A$
 40 IF A$="N" THEN GOTO 120
 50 INPUT"Date de la depense ";D$
 60 INPUT"Nature de la depense";E$
 70 INPUT"Montant de la depense";A
 80 PRINT:PRINT"REGLEZ VOUS CETTE DEPENS
E (O/N) ?":GET A$
 90 IF A$="O" THEN M=M+A:ºWRITE 2'D$,E$,
A
 100 IF A$="N"THEN ºWRITE1'D$,E$,A
 110 PRINT:PRINT:PRINT:GOTO30
 120 ºCLOSE 0
 130 PRINT:PRINT:PRINT:PRINT"LE MONTANT
TOTAL A REGLER SE MONTE A:";M;"Francs."
```

Les dépenses sont alors aiguillées soit vers « FACTURES » si elles sont réglées dans l'immédiat, auquel cas la variable M tient le compte de la somme globale à régler, soit vers « DEPENSES » si elles sont en suspens pour une raison quelconque.

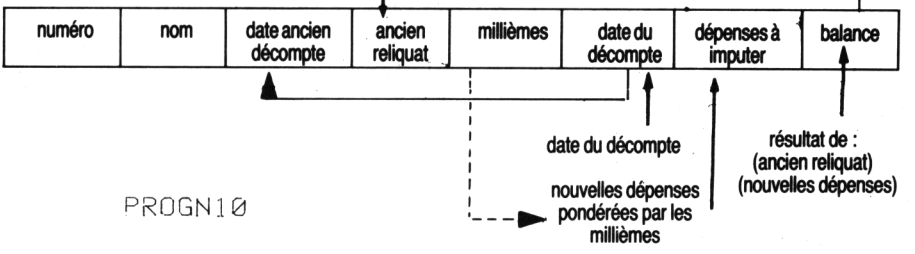
Le montant total à régler à la suite de cette opération est fourni par le programme.

```
 10 REMExamen des depenses et des factur
es
 20 ?OPEN"DEPENSES=1"
 30 ?OPEN"FACTURES=2"
 43 ?REWIND1
 46 ?REWIND2
 50 ?WHERE1
 60 D=PEEK(#48F)-1
 70 ?WHERE2
 80 F=PEEK(#48F)-1
 90 PRINT"Les depenses a la charge de la
 copropriete sont les suivantes:"
 100 PRINT:PRINT:PRINT
 105 D$="":E$=""
 110 FORN=1TO(D/3)
 120 ?TAKE1'D$,E$,A
 130 PRINT"Montant.........";A;"FR."
 140 PRINT"Date............";D$
 150 PRINT"Nature..........";E$
 160 PRINT:PRINT:PRINT
 170 NEXTN
 180 PRINT"Les depenses reglees par la c
opropriete sont les suivantes:"
 190 PRINT:PRINT:PRINT
 200 FORN=1TO(F/3)
 210 ?TAKE2'D$,E$,A
 220 PRINT"Montant.........";A;"FR."
 230 PRINT"Date............";D$
 240 PRINT"Nature..........";E$
 250 PRINT:PRINT:PRINT
 260 NEXTN
 270 ?CLOSE0
 280 STOP
```

Il est possible d'examiner le contenu de ces 2 fichiers à l'aide du programme ci-dessus.

```
Les depenses a la charge de la coproprie
te sont les suivantes:


Montant......... 1700 FR.
Date............03/04/84
Nature..........ELECTRICITE


Montant......... 1300 FR.
Date............18/06/84
Nature..........GARAGE


Les depenses reglees par la copropriete
sont les suivantes:


Montant......... 700 FR.
Date............25/04/84
Nature..........EAU


Montant......... 2500 FR.
Date............12/05/84
Nature..........IMPOTS
```

Dans le cas présent, la somme à régler et donc à répartir entre les copropriétaires se montent à 3.200 F. Il faut donc maintenant faire appel au fichier « SYNDIC », chercher pour chaque copropriétaire le nombre de millièmes qu'il possède et lui imputer la somme correspondante. Puis, en fonction du reliquat du relevé précédent, établir sa nouvelle balance. Ainsi, il faudra appeler chaque enregistrement dans sa totalité puis le réécrire en tenant compte des nouvelles données.



PROGN10

```
10 REM Repartition des charges
15 INPUT"Quel jour sommes nous";C$
20 INPUT"QUELLE LA SOMME A REGLER";A
30 ?OPEN"SYNDIC2=1"
40 ?REWIND1
50 ?TAKE1'G
55 A$="":B$="":D$=""
60 FOR N=1 TO G
70 ?TAKE1'U,A$,B$,SR,M,D$,SD,B
80 B$=D$:D$=C$:SR=B:SD=A*M/1000
90 S=B-SD
110 ?WHERE1
120 O=PEEK(#48D)-7
130 ?REWIND1
140 ?JUMP1,O
145 PRINTU,SD,SR,S
150 ?WRITE1'B$,SR,M,D$,SD,S
180 NEXTN
190 ?CLOSE1
200 STOP
```

Ce programme effectue donc la mise à jour du fichier « SYNDIC ».

Nom du coproprietaire.....DUPONT
Date du dernier decompte..01/01/84
Reliquat a cette date.....-500 Fr.
Milliemes possedes..... 125
A la date du 01/07/84
Les nouvelles depenses ... 400 FR.
conduisent a un SOLDE de.....-900 Fr.

Nom du coproprietaire.....DURAND
Date du dernier decompte..01/01/84
Reliquat a cette date..... 1000 Fr.
Milliemes possedes..... 180
A la date du 01/07/84
Les nouvelles depenses ... 576 FR.
conduisent a un SOLDE de..... 424 Fr.

Nom du coproprietaire.....BRUN
Date du dernier decompte..01/01/84
Reliquat a cette date..... 300 Fr.
Milliemes possedes..... 295
A la date du 01/07/84
Les nouvelles depenses ... 944 FR.
conduisent a un SOLDE de.....-644 Fr.

Nom du coproprietaire.....BELLO
Date du dernier decompte..01/01/84
Reliquat a cette date.....-800 Fr.
Milliemes possedes..... 250
A la date du 01/07/84
Les nouvelles depenses ... 800 FR.
conduisent a un SOLDE de.....-1600 Fr

```
Nom du coproprietaire........DRENO
Date du dernier decompte..01/01/84
Reliquat a cette date..... 700 Fr.
Milliemes possedes........ 150
A la date du .............01/07/84
Les nouvelles depenses ... 480 FR.
conduisent a un SOLDE de........ 220 Fr.
```

Il est possible de vérifier que les 3.200 F en question ont bien été répartis entre les copropriétaires, en fonction de leurs millièmes respectifs, en lançant « PROGN 2 ».

Une fois la mise à jour de « syndic » effectuée, il nous faut prévenir individuellement chaque copropriétaire de l'état de son compte et faire un appel de fonds auprès de ceux pour lesquels celui-ci apparait être débiteur, d'où le programme :

PROGN8

```
 10 REM Appel de fonds
 20 INPUT"QUEL JOUR SOMMES NOUS";E$
 30 PRINT:PRINT:A$="":B$="":C$=""
 40 ?OPEN"SYNDIC2=1"
 50 ?REWIND1
 60 ?TAKE1'G
 70 FOR N =1 TO G
 80 ?TAKE1'U,A$,B$,SR,M,C$,SD,B
 90 IFB<0THENGOSUB400ELSEGOSUB800
 100 NEXTN
 110 ?CLOSE0
 120 STOP
 400 REM Lettre pour les debiteurs
 410 LPRINT"Monsieur ";A$:LPRINTTAB(35)"
Marseille,le ";E$
 420 LPRINT"Residence FLORIDA no";U
```

```
430 LPRINT:LPRINT:LPRINT:LPRINT
440 LPRINT"                    Monsieur,"
450 LPRINT:LPRINT"          L'examen semes
triel des comptes de votre ";
460 LPRINT"copropriete au ";C$;" laisse
apparaitre a votre debit la somme de";
470 LPRINT" ";-B;"Fr.Je vous serais gre
de bien vouloir la couvrir dans les";
480 LPRINT" meilleurs delais."
490 LPRINT:LPRINT:LPRINT"        Je vous
prie d'agreer,Monsieur,l'exp......"
500 FORJ=1TO10:LPRINT:NEXTJ
510 RETURN
800 REM Lettre pour les autres
810 LPRINT"Monsieur ";A$:LPRINTTAB(35)"
Marseille,le ";E$
820 LPRINT"Residence FLORIDA no";U
830 LPRINT:LPRINT:LPRINT:LPRINT
840 LPRINT"                    Monsieur,"
850 LPRINT:LPRINT"          L'examen semes
triel des comptes de votre ";
860 LPRINT"copropriete laisse apparaitr
e a votre credit la somme de";B;"Fr."
870 LPRINT"       En consequence,aucun ap
pel de fonds ne sera effectue ce trim";
880 LPRINT"estre."
890 LPRINT:LPRINT:LPRINT"        Je vous
prie d'agreer,Monsieur,l'exp......"
900 FORJ=1TO10:LPRINT:NEXTJ
910 RETURN
```

Monsieur DUPONT

Marseille, le 02/09/84

Residence FLORIDA no 1

Monsieur,

L'examen semestriel des comptes de votre copropriete au 01/07/84 laisse apparaitre a votre debit la somme de 900 Fr. Je vous serais gre de bien vouloir la couvrir dans les meilleurs delais.

Je vous prie d'agreer, Monsieur, l'exp.....

Monsieur DURAND

Marseille, le 02/09/84

Residence FLORIDA no 2

Monsieur,

L'examen semestriel des comptes de votre copropriete laisse apparaitre a votre credit la somme de 424 Fr.

En consequence, aucun appel de fonds ne sera effectue ce trimestre.

Je vous prie d'agreer, Monsieur, l'exp.....

Monsieur BRUN

Marseille, le 02/09/84

Residence FLORIDA no 3

Monsieur,

L'examen semestriel des comptes de votre copropriete au 01/07/84 laisse apparaitre a votre debit la somme de 644 Fr. Je vous serais gre de bien vouloir la couvrir dans les meilleurs delais.

Je vous prie d'agreer, Monsieur, l'exp.....

Monsieur BELLO

Marseille, le 02/09/84

Residence FLORIDA no 4

Monsieur,

L'examen semestriel des comptes de votre copropriete au 01/07/84 laisse apparaitre a votre debit la somme de 1600 Fr. Je vous serais gre de bien vouloir la couvrir dans les meilleurs delais.

Je vous prie d'agreer, Monsieur, l'exp.....

```
Monsieur DRENO
                                Marseille,le 02/09/84
Residence FLORIDA no 5



                    Monsieur,

          L'examen semestriel des comptes d
e votre copropriete laisse apparaitre a
votre credit la somme de 220 Fr.
       En consequence,aucun appel de fonds
 ne sera effectue ce trimestre.


          Je vous prie d'agreer,Monsieur,l'e
xp......
```

Cependant, certaines dépenses ont été laissés en suspens (litiges...). On peut en examiner la liste de temps à autre, et éventuellement se décider à régler certaines d'entre elles. D'où le programme.

```
PROGN11

 10 REM Reglement des depenses en suspen
s
 20 !CREATE"DEPENSE1,S=3"
 30 !OPEN"DEPENSES=1"
 40 !OPEN"FACTURES=2"
 45 D$="":E$="":M=0
 50 !APND2
 60 !REWIND1
 70 !WHERE1
 80 A=((PEEK(#48F)-1)/3)
```

```
 90 FORN=1TOA
 100 !TAKE1'D$,E$,S
 110 PRINTD$,E$,S;"Fr."
 120 PRINT"Voulez vous regler cette depe
nse?"
 130 GETB$
 140 IF B$="O"THEN GOTO300
 150 IF B$="N"THEN GOTO400
 160 GOTO 120
 170 NEXTN
 180 PRINT"Le montant a regler est de ";
M;"Fr."
 190 !CLOSE0
 200 !DEL"DEPENSES.DAT"
 210 !RENAME"DEPENSE1.DAT=DEPENSES.DAT"
 220 STOP
 300 REM Passage DEPENSES -> FACTURES
 305 M=M+S
 310 !WRITE2'D$,E$,S
 320 GOTO170
 400 REM Passage DEPENSES -> DEPENSE1
 410 !WRITE3'D$,E$,S
 420 GOTO 170
```

Celui-ci fait intervenir 3 fichiers : DEPENSES et FACTURES, déjà connus et un nouveau fichier tampon : DEPENSE 1.

DEPENSES est examiné à partir du début (par ! REWIND) séquentiellement. Deux solutions existent pour chaque enregistrement :

- soit l'on décide de régler la dépense, auquel cas l'enregistrement est transféré dans FACTURES,

- soit l'on persiste à la laisser en suspens ; l'enregistrement est alors transféré dans DEPENSE 1.

L'enregistrement est alors transféré dans DEPENSE 1. Une fois tout le fichier examiné, DEPENSES est détruit, pour être remplacé par DEPENSE 1, qui prend alors son nom.

```
Les depenses a la charge de la coproprie
te sont les suivantes:


Montant......... 1300 FR.
Date............18/06/84
Nature..........GARAGE


Les depenses reglees par la copropriete
sont les suivantes:


Montant......... 700 FR.
Date............25/04/84
Nature..........EAU


Montant......... 2500 FR.
Date............12/05/84
Nature..........IMPOTS


Montant......... 1700 FR.
Date............03/04/84
Nature..........ELECTRICITE
```

Dans notre cas, nous décidons de régler le poste ELECTRICITE. Un examen de DEPENSES et FACTURES par PROGN 5 nous confirme que tout s'est bien passé.

LES FICHIERS A ACCES DIRECT

INTRODUCTION

Ils sont parfois appelés aussi fichier à accès aléatoire. Cela ne veut pas dire que retrouver une information tient du hasard (bien que dans certains cas... !), mais que pour accéder à l'information souhaitée on n'a pas besoin de connaître toutes les informations précédantes.

Les fichiers à accès séquentiels sont relativement faciles à mettre en place et à consulter. D'autant plus que le DOS offre des instructions puissantes de maniement. Néanmoins la recherche d'un enregistrement précis est relativement longue, la mise à jour du fichier nécessite généralement la recopie dans un autre fichier.

Ces inconvénients sont partiellement gommés avec les fichiers à accès direct. L'accès direct est permis avec un lecteur de disquette car la tête a la possibilité d'accéder à n'importe qu'elle piste. (En réalité on n'accède donc pas directement à un enregistrement mais à une piste, ensuite une lecture séquentielle des secteurs de cette piste permet de trouver l'enregistrement, cette manœuvre restant totalement transparente pour l'utilisateur car c'est le DOS qui se préoccupe de la recherche). Notons cependant que le DOS doit connaître les pistes sur lesquelles est sauvé le fichier et doit aussi connaître le nombre d'enregistrement par piste pour pouvoir calculer le numéro de la piste où se trouve un enregistrement donné. Cette dernière contrainte impose des enregistrements de longueurs fixe et constante à l'intérieur d'un même fichier.

Le problème avec un fichier à accès direct est que pour accéder à un enregistrement il faut connaître son numéro d'enregistrement, ce qui n'est pas toujours le cas, dès ce moment on n'est plus dans les conditions de l'accès direct et il faut parcourir le fichier séquentiellement (on a rien gagné et on a perdu de la place sur le disque). Nous verrons quelques méthodes d'accès à un enregistrement sans connaître son numéro.

Le lecteur de disquette a 3 instructions spéciales pour les fichiers à accès direct. ! CREATE, ! WRITE, ! TAKE.

Pour utiliser un fichier à accès direct la procédure est la même que pour un fichier à accès séquentiel, il faut d'abord le déclarer, le créer ou l'ouvrir, puis on peut lire les informations, écrire de nouvelles infos, ensuite il faut le refermer.

⊗ ! CREATE « (numéro de disque.) **nom du fichier D = NLU, LR, NE**»

NLU est le numéro logique affecté au fichier. Il a la même signification que pour un fichier à accès séquentiel. NLU doit être compris entre 1 et 255 ; Ø n'est pas autorisé, NLU peut être un nombre hexadécimal ou même une variable numérique.

LR : longueur d'enregistrement, elle est comptée en octets, LR inférieure à 255.

L'enregistrement à une longueur fixe celle que vous allez déclarer, les rubriques ont, elles, une longueur variable qui est de toute façon incluse ou égale à LR. Il va falloir que vous évaluiez cette longueur. Pour cela, il faut connaître qu'elles sont les rubriques qui formeront le fichier. Exemple : imaginons que l'on veuille mettre sur un fichier un carnet d'adresse.

Une adresse c'est un nom + un prénom + une rue + n° de rue + code postal + ville + n° de téléphone.
DURAND Jacques
30 rue Bizuz
13410 MARSEILLE 16 (91) 99.33.20
Pour chaque champ on va définir la place occupée sachant qu'une variable entière occupe 8 octets,
- une variable réelle occupe 17 octets,
- une variable alpha occupe 1 octet par caractère dans la chaîne + 2 octets.

Pour notre exemple :
nom = 10 caractères + 2 Alpha
prénom = 10 caractères + 2 Alpha
n° rue = entier 8 octets

nom rue = 50 caractères + 2 Alpha
code postal = entier 8 octets
ville = 20 caractères + 2 Alpha
n° téléphone =12 caractères + 2 Alpha

Dans cet exemple :
LR = 10 + 2 + 10 + 2 + 8 + 50 + 2 + 8 + 20 + 2 + 12 + 2.
LR =128 octets.

NE : nombre d'enregistrements réservés à l'avance, il conditionne la place réservée sur le disque. Il faut le gérer en fonction de l'étendue prévue du fichier. Ce n'est pas toujours très commode de prévoir à l'avance l'importance du fichier. Dans le cas où le fichier source devient trop petit on peut s'en sortir en le recopiant séquentiellement dans un autre fichier plus grand.

Dans notre exemple, la place pour 200 fiches (200 adresses complètes) occupe 103 secteurs. En effet, chaque secteur est constitué de 256 octets donc il va 2 enregistrements par secteur (200/2 = 100 secteurs) les 3 secteurs restant sont utilisés pour le descriptif au fichier : ils ne sont pas accessibles à l'utilisateur.

Nous allons créer un fichier pour notre répertoire téléphonique
! CREATE « AGENDA, D = 1, 128, 200 »

La commande ! CAT «» nous permet de vérifier l'effet de ! CREATE. Maintenant dans la liste des fichiers apparaît :
U AGENDA .DAT D 103 SECTORS.
En clair AGENDA est un fichier direct (D), donc de type DATA (DAT) il n'est pas protégé en écriture/effacement (U), il occupe sur la disquette 103 secteurs.

- l'utilisation de la fonction ! CREATE «...» pour un fichier qui existe déjà conduit au message d'erreur
EXISTING FILE !
- vous ne pouvez pas non plus utiliser un fichier à accès séquentiel de même nom.

⊗ ! OPEN « (numéro de disque.) nom de fichier = NLU »

Cette instruction permet de travailler dans un fichier qui existe déjà. Elle signale au DOS de sortir le « dossier » nom de fichier, de le placer à l'endroit désigné par le nombre NLU. En termes plus informatiques cette instruction vous permet d'accéder au fichier en lui attribuant un numéro NLU. NLU peut être explicite dans l'instruction ou représentée par une variable. NLU doit être 1≤NLU≤255. Ø n'est pas autorisé. Une tenta-

tive d'ouverture d'un fichier qui n'a pas encore été créé conduit immanquablement au message d'erreur n° 7 : FILE NOT FOUND. L'écriture ou la lecture d'un fichier fermé produit le même message d'erreur.

⊗ ! WRITE NLU, E' A,B,C,D%,K%,A$

Cette instruction permet d'écrire dans le fichier NLU, à l'enregistrement E, les variables réelles A,B,C, les variables entières D%, K% et la variable alphanumérique A$ sous réserve que la longueur d'enregistrement prévue soit supérieure ou égale à (17 x 3 + 8 x 2 + 2 + nombre caract A$). Les rubriques sont enregistrées dans cet ordre : il ne s'agira pas de les lire dans n'importe quel ordre car sinon le résultat obtenu sera incohérent.

Tous les types de variables sont acceptés sauf les tableaux de chaînes de caractères et les tableaux entiers ou réels de plus d'une dimension.

A$ (I)	A (I)
C% (I,I)	B% (I)
Mauvais	Bon

Que se passe-t-il si un enregistrement est plus long que le nombre d'octets autorisés au moment de la création du fichier ? Dans ce cas le DOS détecte une erreur et renvoie le message : OUT OF DATA (n° 16). Ce qui signifie qu'on a essayé d'écrire sur un enregistrement voisin. L'enregistrement sur lequel on a tenté de mordre n'est cependant pas détruit.

E est le numéro qui permet d'écrire dans un enregistrement particulier. En effet, rappelons-le dans un fichier à accès direct les enregistrements ne sont pas obligatoirement contigus. E permet d'adresser chacun d'eux. E doit être compris entre 1≤E≤65535 ou # FFFF ; il peut être représenté par une variable. E ne doit pas être plus grand que le nombre maxi de fiches sinon le message « END OF FILE » est généré par le DOS.

Remarques très importantes : le nom d'une variable alphanumérique ne doit en aucun cas être suivi par un espace.
! WRITE 1, 2', A$△, B ou ! WRITE 1,2'B,A$△ sont des expressions erronnées si △ représente un espace.
! WRITE 1,2' A$,B et ! WRITE 1,2'B,A$ sont correctes. L'espace conduit au message : SYNTAX ERROR, et comme il est invisible c'est une erreur difficile à corriger surtout en fin de ligne donc Attention !

Une variable chaîne de caractères doit être initialisée avant d'être utilisée. Autrement dit il faut déclarer dans le programme toutes les variables alphanumériques utilisées . Par exemple :

```
10 A$=«»
20 OPEN «TEL=1»
30 ! WRITE 1,1,A$
```

si on omet la ligne 10, RUN est suivi par MISSING VARIABLE message d'erreur nº 14.

Lecture dans un fichier à accès direct.

! TAKE NLU, E' A,B,C$,D%,J(I)

Permet de saisir dans l'enregistrement nº E du fichier nº NLU les rubriques et de les affecter aux variables réelles A,B aux variables alphanumérique C$, entière D% et réelle en tableau à 1 dimension J. Naturellement le type des variables enregistrées doit correspondre au type des variables lues. S'il n'y a pas concordance le message « ? TYPE MISMATCH ERROR » est édité.

* E représente le numéro d'enregistrement, il doit être compris entre 1≤ E≤65535 ou # FFFF ; il peut être contenu dans une variable. C'est grâce à ce nombre que l'on peut accéder à un enregistrement particulier dans le fichier.

Les remarques formulées au sujet de l'instruction ! WRITE restent valables : une chaîne de caractère doit être déclarée avant d'être utilisée.

- Il ne doit pas y avoir d'espace après une variable chaîne de caractère.

! CLOSE NLU ou ! CLOSE Ø

Permettent (comme dans le cas de fichiers à accès séquentiels) respectivement de fermer le fichier de numéro logique NLU et de fermer tous les fichiers en cours.

! CLOSE 2 ferme le fichier référencé 2.

Il est bien sûr impossible de lire ou d'écrire dans un fichier fermé (de la même façon que vous ne pouvez ni lire, ni écrire au travers d'une pochette ou d'un dossier fermé).

⊗ ! WHERE NLU et ! APND NLU

Notons pour mémoire que ces 2 instructions bien que réservées aux fichiers à accès séquentiels fonctionnent pour les fichiers directs, leur intérêt n'est pas très grand ici.

! WHERE NLU fourni à l'adresse = 48 F le nombre d'enregistrements réservés pour le fichier lors de l'instruction ! CREATE «» par exemple :
! CREATE « ESSAI, D = 1, 100,50 » ↵
! WHERE 1 ↵ et PRINT PEEK (# 48 F) affiche 50

⊗ ! WL NLU et WUL NLU

! WL NLU : (Write Lock) permet d'empêcher l'écriture dans un fichier déjà ouvert. Cette instruction est pratique surtout pendant la phase de mise au point d'un programme. Ainsi, elle évite de détruire par erreur les enregistrements d'un fichier précieux.
! WUL NLU : est la commande inverse.

Attention : ! WL NLU est moins puissante que ! LOCK «..» elle ne protège pas le fichier contre ! DEL, ! SAVE.

Maintenant que nous avons vu les principales instructions d'un fichier à accès direct nous allons les manipuler pour nous familiariser.

Ce programme est très simple au début, ensuite nous allons le peaufiner un peu au fur et à mesure que nous prendrons plus d'assurance avec ce type de fichiers.

Pour l'instant définissons les rubriques de notre fichier. On va utiliser celles du fichier défini au paragraphe ! CREATE, on va réserver la place pour 10 fiches. Supposons dans un premier temps que le numéro d'enregistrement sera donné de façon arbitraire ; dans l'ordre où on pense aux personnes on leur attribue leur numéro.

1° - CREATION DU FICHIER

On utilise simplement l'instruction.
! CREATE « AGENDA, D = 1, 128, 10 » ↵ (↵ = return)
un coup d'œil au catalogue montre
U AGENDA • DAT D 8 SECTORS.
Notre fichier occupe 5 secteurs pour les fiches (10 x 256 : 128)
+ 3 secteurs pour le DOS.

2° - SOUS PROGRAMME DE REMPLISSAGE DU FICHIER

Commentaires :
La ligne 50 affecte au fichier « agenda » précédemment crée le numéro logique 1, donc chaque fois que nous ferons référence au fichier n° 1 nous

nous adresserons à « agenda ».
60 : NE représente le nombre d'enregistrements pour lequel a été dimensionné le fichier.
70 : EC = enregistrement courant. C'est le compteur d'enregistrement il permet d'écrire dans le fichier. EC est incrementé de 1 à chaque tour de boucle (240). S'il devient plus grand que NE on sort de la boucle (270).

```
REMPLISS

 1 CLS
 10 '*******************************
 20 '*       OUVERTURE DU FICHIER    *
 30 '*******************************
 40 PRINT "FICHIER AGENDA"
 50 !OPEN"AGENDA=1"
 60 NE=10
 70 EC=1
 80 '
 90 ' ******************************
 100 '*    REMPLISSAGE DU FICHIER   *
 110 '******************************
 120 '
 130 REPEAT
 140 INPUT"NOM ";NO$
 150 INPUT"PRENOM ";PN$
 160 INPUT"NUMERO DE RUE ";NR%
 170 INPUT"ADRESSE ";AD$
 180 INPUT"CODE POSTAL ";CP
 190 INPUT"VILLE ";VI$
 200 INPUT"NUMERO DE TELEPHONE ";TE$
 210 '
 220 !WRITE1,EC'NO$,PN$,NR%,AD$,CP,VI$,T
E$
 230 '
 240 EC=EC+1
 250 PRINT "ENCORE UN ENREGISTREMENT ?"
 260 GET A$
```

```
270 UNTIL EC=11 OR A$<>"O"
280 '
300 '*******************************
310 '*   FERMETURE DU FICHIER      *
320 '*******************************
330 '
340 !CLOSE1
350 END
```

```
EXEMPLE D'EXECUTION

FICHIER AGENDA
NOM ? ASPER
PRENOM ? JULES
NUMERO DE RUE ? 12
ADRESSE ? RUE BLEUE
CODE POSTAL ? 84000
VILLE ? AVIGNON
NUMERO DE TELEPHONE ? 90 87 23 15
ENCORE UN ENREGISTREMENT ?
```

130-200 : Affectation des variables ;

On remarque simplement que le code postal occupe une variable réelle, pourquoi ? bien que le code postal soit toujours un entier, une variable entière ne convient pas car elle ne peut contenir que des nombres ‹ 32768.

220 : C'est cette ligne qui écrit dans le fichier, les rubriques sont placées dans l'ordre NOM, PRENOM, NUMERO DE RUE, ADRESSE, CODE POSTAL, VILLE, NUMERO DE TELPHONE.
340 : fermeture du fichier, ! CLOSE Ø aurait aussi convenu.

3° - SOUS PROGRAMME DE LECTURE DANS LE FICHIER.

Maintenant nous allons récupérer les informations emmagasinées dans le fichier. (voir programme lecture d'un fichier).

Commentaires :
50 : ouverture du fichier cette manœuvre est systématique
60 : NE nombre total d'enregistrements du fichier.
70-80 : Déclaration des variables qui récupèrent les différentes rubriques d'un enregistrement, cette phase est absolument nécessaire uniquement pour les variables alphanumériques.
130-140 : Choix de l'enregistrement à lire ; Ici est la grande différence entre les fichiers à accès direct et séquentiel. Vous pouvez lire n'importe quel enregistrement dans un ordre quelconque.
145 : vérification que le numéro donné est bien dans les limites du fichier.
150 : lecture de l'enregistrement dans le fichier.

Lecture d'un fichier.

```
LECAG

 1 CLS
 10 '*******************************
 20 '*        OUVERTURE DU FICHIER   *
 30 '*******************************
 35 '
 40 PRINT "LECTURE DU FICHIER AGENDA"
 50 ?OPEN "AGENDA=1"
 60 NE=10
 70 NO$="":PN$="":NR%=0:AD$=""
 80 CP=0:VI$="":TE$=""
 90 '
 100 '*******************************
 110 '*    LECTURE DANS LE FICHIER   *
 120 '*******************************
 125 '
 130 PRINT "DONNEZ LE NUMERO D'ENREGISTR
EMENT";
 140 INPUT NE
 150 ?TAKE1,NE'NO$,PN$,NR%,AD$,CP,VI$,TE
$
 160 PRINT "NOM :";NO$;"  ";PN$
 170 PRINT "ADRESSE :"
```

```
180 PRINT NR%;" ";AD$
190 PRINT "     ";CP;"     ";VI$
200 PRINT"NUMERO DE TELEPHONE : ";TE$
210 PRINT
220 PRINT "UN AUTRE ? (O/N)  ";
230 GET A$
240 IF A$="O"THEN GOTO 130
250 IF A$="N"THEN CONT ELSE GOTO 230
300 '*****************************
310 '*   FERMETURE DU FICHIER     *
320 '*****************************
330 '
340 ?CLOSE0
350 END
```

```
LECTURE DU FICHIER AGENDA
DONNEZ LE NUMERO D'ENREGISTREMENT 1
NOM :DUPONT  JACQUES
ADRESSE :
 19  CHEMIN DES SALINS
     13012     MARSEILLE
NUMERO DE TELEPHONE : 91 42 35 87

UN AUTRE ? (O/N)
```

160-200 : Mise en forme du texte à afficher sur l'écran.
250 : lecture d'un autre enregistrement ?
340 : si non fermeture du fichier.

4° - Vous venez de rencontrer de nouveaux amis et vous voulez mettre à jour votre fichier. Le programme suivant vous en donne la possibilité. Mais auparavant il se pose un problème comment savoir si un enregistrement est libre puisque nous ne sommes pas obligés d'écrire les enregistrements les uns à la suite des autres.

Nous voyons apparaître une contrainte supplémentaire : pouvoir distinguer un enregistrement vide d'un occupé. La solution à ce problème la plus souvent employée est de remplir les enregistrements d'un caractère nul pendant l'initialisation.

```
INITFICH

 10 '*******************************
 20 '*   CREATION D'UN FICHIER      *
 30 '*******************************
 40 '
 50 INPUT "NOM DU FICHIER ";N$
 60 INPUT "NOMBRE MAXI D'ENREGISTREMENTS
 ";N
 70 INPUT "LONGUEUR D'UN ENREGISTREMENT
";LR
 75 N$=N$+",D=1,"+STR$(LR)+","+STR$(N)
 80 !CREATE N$
 90 '
 100 '*******************************
 110 '*  REMPLIR LE FICHIER         *
 120 '*******************************
 130 A$=CHR$(2)
 140 FOR I=1 TO N
 150 !WRITE1,I'A$
 160 NEXT I
 170 !CLOSE1
 180 END


NOM DU FICHIER ? AGENDA1
NOMBRE MAXI D'ENREGISTREMENTS ? 10
LONGUEUR MAXI D'UN ENREGISTREMENT ? 128
```

La ligne 70 permet de fabriquer la chaîne N$ adéquate pour l'instruction create.
130 : initialise A$ avec caractère nul (code ASC II)

140 à 160 : écriture dans tout le fichier de A$.
Maintenant, l'insertion d'un élément devient on ne peut plus simple. (programme insère)
Commentaires :
130-190 : Entrée des coordonnées d'une fiche
200 : Entrée du numéro de fiche où seront affectées les données précédentes.
240-270 : vérification que cette fiche est libre, on lit la 1re rubrique de la fiche : si elle est vide A$ contient le caractère de code ASC II = 2, si elle est occupée A$ contient un nom.

INSERE

```
1 CLS
10 '*******************************
20 '*    OUVERTURE DU FICHIER      *
30 '*******************************
40 '
50 ?OPEN"AGENDA=1"
60 NE=10:A$=""
100 '*******************************
110 '*    INSERTION D'UN ELEMENT    *
120 '*******************************
130 INPUT"NOM ";NO$
140 INPUT"PRENOM$ ";PN$
150 INPUT"NUMERO DE RUE ";NR%
160 INPUT"ADRESSE ";AD$
170 INPUT"CODE POSTAL ";CP
180 INPUT"VILLE ";VI$
190 INPUT"NUMERO DE TELEPHONE ";TE$
200 INPUT "NUMERO D'ENREGISTREMENT ";EC
210 '*******************************
220 '*   VERIFICATION              *
230 '*******************************
240 '
250 ?TAKE1,EC'A$
260 IF A$<>CHR$(2) THEN GOTO 200
```

```
270 ?WRITE1,EC'NO$,PN$,NR%,AD$,CP,VI$,T
E$
280 PRINT "UNE AUTRE INSERTION "
290 GET B$
300 IF B$="O"THEN GOTO 130
310 '
320 '*****************************
330 '*     FERMETURE FICHIER       *
340 '*****************************
350 '
360 ?CLOSE1
370 END
```

Si la fiche est libre on écrit le nouvel enregistrement. Dans le cas contraire, un autre numéro de fiche est demandé sa validité est à nouveau vérifiée.

Notons tout de suite une amélioration possible : si le numéro entré est supérieur au nombre total de fiches que peut contenir le fichier le DOS génèrera l'erreur END OF FILE est le programme se bloquera obligeant l'opérateur à réentrer toutes les données.

Pour palier à cet inconvénient, on peut rajouter la ligne 205 IF EC › NE THEN GOTO 200.

5° - SUPPRESSION D'UN ENREGISTREMENT.

Vous vous êtes fachés avec quelques relations, supprimez-les donc du fichier. La suppression d'un enregistrement revient ici à écrire un caractère de code ASCII = 2 dans la première rubrique ; ainsi lors de l'insertion d'un autre élément cet enregistrement sera évalué libre. Contrairement aux fichiers à accès séquentiel qui demandent généralement la recopie totale du fichier, les fichiers à accès direct permettent les suppressions de façon simple et quasi instantanée.

Commentaires :

Les lignes 1 à 60 vous sont désormais familières.

70 : initialisation de la variable A$ avec le caractère de code ASCII 2.

130 : demande de numéro de l'enregistrement à effacer.

140 : cette ligne est une sécurité au cas où l'opérateur aurait tapé un mau-

vais nº d'enregistrement. Dans ce cas un autre numéro est demandé. La fin du programme vous est aussi habituelle : (demande si un autre enregistrement doit être effacé, ferme le fichier).

```
SUPPRES

 1 CLS
 10 '*******************************
 20 '*   OUVERTURE DU FICHIER      *
 30 '*******************************
 35 '
 40 ¤OPEN"AGENDA=1"
 45-PRINT"FICHIER AGENDA":PRINT:PRINT
 50 NE=10
 60 EC=1
 70 A$=CHR$(2)
 80 '
 90 ' ******************************
 100 '*        SUPPRESSION          *
 110 '******************************
 120 '
 130 INPUT "NUMERO D'ENREGISTREMENT A SU
PPRIMER";EC
 140 PRINT CHR$(27);"AOK....? (O/N)"
 150 GET B$
 160 IF B$<>"O"AND B$<>"N" THEN GOTO 150
 170 IF B$<>"O" THEN GOTO 130
 180 ¤WRITE1,EC'A$
 190 PRINT"UN AUTRE ? (O/N)"
 200 GET B$
 210 IF B$="O" THEN GOTO130
```

```
230 '*******************************
240 '*     FERMETURE DU FICHIER     *
250 '*******************************
260 '
270 !CLOSE1
280 END
```

```
EXEMPLE D'EXECUTION

FICHIER AGENDA
NUMERO D'ENREGISTREMENT A SUPPRIMER: 1
OK....?(O/N)  O
UN AUTRE ? (O/N)
```

6° - La dernière opération importante sur un fichier à accès direct, est de pouvoir lister de façon séquentielle tous les enregistrements existants. Le programme présenté ci-dessous autorise cette opération.

Commentaires :
130-140 : Notons ici l'utilisation de l'instruction ! WHERE 1 associée à peek (# 48 F) qui met dans la variable NE le nombre total d'enregistrement du fichier.
152-160 : on lit le premier champ de chaque enregistrement, s'il est égal à CHR$ (2) alors on le saute, sinon c'est un enregistrement qui existe donc on peut lire les champs suivants ; remarquons que contrairement à un fichier séquentiel on relit la totalité de l'enregistrement (NO$ correspond alors à A$).
170-210 : Impression des résultats sur l'imprimante.
280 : Fermeture du fichier.

```
LISTESEQ

1 CLS
10 '*******************************
20 '*    OUVERTURE DU FICHIER      *
30 '*******************************
35 '
40 !OPEN"AGENDA=1"
45 PRINT"FICHIER AGENDA":PRINT:PRINT
```

```
50 NO$="":PR$="":NR%=0:AD$="":CP=0:VI$=
"":TE$=""
60 EC=1
70 A$=CHR$(2)
80 '
90 ' ******************************
100 '*      LISTE SEQUENTIELLE       *
110 '******************************
120 '
130 ?WHERE1
140 NE=PEEK(#48F)
150 FOR I=1 TO NE
152 ?TAKE1,I'A$
155 IF A$=CHR$(2) THEN GOTO 220
160 ?TAKE1,I'NO$,PR$,NR%,AD$,CP,VI$,TE$
170 LPRINT"ENREGISTREMENT NUMERO ";I
180 LPRINT NO$;"    ";PR$
190 LPRINT NR%;" ";AD$
200 LPRINT CP;"  ";VI$
210 LPRINT TE$
220 NEXT I
230 '
240 '******************************
250 '*          FERMETURE           *
260 '******************************
270 '
280 ?CLOSE1
290 END
```

```
ENREGISTREMENT NUMERO  1
DUPONT    LUC
 24  RUE DES SALINS
 13004    MARSEILLE
91 74 34 23
ENREGISTREMENT NUMERO  2
```

```
ANETON    ALAIN
 19  AVENUE JAUNE
 12000    LE LUC
23 45 67 43
ENREGISTREMENT NUMERO  8
SARTIN    PIERRE
 56  AVENUE DE TOULON
 83000    TOULON
34 23 12
```

7° - RECHERCHE D'UN ENREGISTREMENT PAR L'INTERMEDIAIRE D'UN NOM

Nous allons voir que cette méthode est guère différente de la liste séquentielle. Avec cette méthode on ne distingue plus très bien l'intérêt du fichier à accès direct, nous verrons plus loin qu'il existe heureusement d'autres méthodes qui rendent l'utilisation d'un fichier direct plus optimale.

Pour l'instant, intéressons-nous au programme, la ligne 130 attend l'entrée d'un nom ;
ensuite lignes 170-180-190. On prend le premier champ de chaque enregistrement, il est comparé au mot-clé contenu dans CL$, s'ils sont différents on lit l'enregistrement suivant sinon un indicateur est mis = 1 (195), l'enregistrement est lu en entier (190) et imprimé.

250 : en sortant de la boucle on teste l'indicateur ; s'il est à 1 le nom cherché existe dans le fichier il a été imprimé, s'il est à zéro le nom cherché n'existe pas on imprime NOM INCONNU.

```
RECSEQCL

 1 CLS
 10 '*******************************
 20 '*   OUVERTURE DU FICHIER      *
 30 '*******************************
 35 '
 40 ?OPEN"AGENDA=1"
 45 PRINT"FICHIER AGENDA":PRINT:PRINT
 50 NO$="":PN$="":NR%=0:AD$="":CP=0:VI$=
```

```
"":TE$=""
 60 INDIC=0
 70 A$=""
 80 '
 90 ' ********************************
 100 '*   RECHERCHE PAR CLE           *
 110 '********************************
 120 '
 130 INPUT"QUEL NOM ";CL$
 140 ?WHERE1
 150 NE=PEEK(#48F)
 160 FOR I=1 TO NE
 170 ?TAKE1,I'A$
 180 IF A$<>CL$ THEN GOTO 240
 190 ?TAKE1,I'NO$,PN$,NR%,AD$,CP,VI$,TE$
 195 INDIC=1
 200 PRINT NO$;"     ";PN$
 210 PRINT NR%;" ";AD$
 220 PRINTCP;" ";VI$
 230 PRINTTE$
 240 NEXT I
 250 IF INDIC<>1 THEN PRINT "NOM INCONNU
"
 260 '
 270 '********************************
 280 '*           FERMETURE           *
 290 '********************************
 300 '
 310 ?CLOSE1
 320 END
```

8° - Maintenant que toutes les routines de base ont été décrites, nous allons donner un programme de synthèse du répertoire d'adresses. Définissons en les buts : ce programme doit permettre de :
- créer un fichier,
- de le mettre à jour (insertion et suppression de fiche) à partir d'un numéro de fiche,
- de lire une fiche à partir d'un numéro, d'un nom, d'un prénom,
- d'éditer la totalité du fichier.

Ce programme a été développé dans plusieurs buts :
- la rapidité d'exécution n'a pas été recherchée, au profit de la clarté du programme,
- le programme est aussi structuré que possible,
- il donne un exemple de masque de saisie données,
- il fait la synthèse des diverses opérations déjà vues sur les fichiers à accès direct.

Il est écrit pour l'atmos ; il fonctionne pour ORIC 1 à condition de simuler PRINT aX,Y;«»;

Par exemple toutes les instructions PRINT a sont remplacées par la ligne X = .. : Y = .. : A$ = «...» : GOSUB 16000

```
 16000 POKE 616, Y
 16010 PRINT
 16020 POKE 617, X
16030 PRINT A$;
16040 RETURN.
```

Commentaires sur le programme : voir après le listing.

AGENDA

```
 10 CLS
 20 NO$="":PN$="":AD$="":VI$="":TE$=""
 30 PRINT@6,14;"SORTIE SUR IMPRIMANTE O/
N? -";CHR$(8);
 40 GET A$
 50 IF A$="N" THEN IM=0:POKE 48020,32
 60 IF A$="O" THEN IM=1:POKE 48020,73
 70 CLS
 80 '
 90 ' ********
 100 '* MENU *
 110 '********
 120 '
 130 PRINT"CREER UN FICHIER = CRE"
 140 PRINT"INSERER UNE FICHE = INS"
 150 PRINT"EFFACER UNE FICHE = EFF"
 160 PRINT"LIRE FICHE A PARTIR D'UN NUME
RO = LNU
 170 PRINT"LIRE FICHE A PARTIR D'UN NOM
= LNO"
 180 PRINT"LIRE FICHE A PARTIR D'UN PREN
OM = LPR"
 190 PRINT"EDITION TOTALE DU FICHIER =ED
I"
 200 PRINT"FIN = FIN"
 210 PRINT@19,15;"---":PRINT@19,15;"";
 220 A=3:B$="":GOSUB 3160
 230 IF LEN(B$)=3 THEN GOTO 250
 240 PING:GOTO 210
 250 IF B$="CRE" THEN GOSUB  370
 260 IF B$="INS" THEN GOSUB  580
 270 IF B$="EFF" THEN GOSUB  970
 280 IF B$="LNU" THEN GOSUB 1280
 290 IF B$="LNO" THEN GOSUB 1550
```

```
300 IF B$="LPR" THEN GOSUB 1780
310 IF B$="EDI" THEN GOSUB 2010
320 IF B$="FIN" THEN POKE 48020,32:END
330 GOTO 70
340 '************************
350 '* CREATION D'UN FICHIER *
360 '************************
370 '
380 CLS:PRINT:PRINT:PRINT:PRINT
390 INPUT"NOM DU FICHIER ";B$
400 IF B$="" THEN RETURN
410 PRINT:PRINT
420 INPUT"NOMBRE MAXI D'ENREGISTREMENTS
";N
430 PRINT:PRINT
440 N$=B$+",D=1,128,"+STR$(N)
450 !CREATE N$
460 GOSUB 3710
470 A$=CHR$(2)
480 FOR I=1 TO N
490 !WRITE1,I'A$,A$
500 NEXT I
510 GOSUB  3500
520 RETURN
530 '
540 '********************
550 '* INSERER UNE FICHE *
560 '********************
570 '
580 GOSUB  3600
590 IF PB=1 THEN RETURN
600 GOSUB 2130
610 CLS
620 IF NO$="" THEN GOTO  790
630 PRINT@7,10;"NUMERO D'ENREGISTREMENT
: -----"
```

```
640 PRINT@10,17;"0 = LE PREMIER LIBRE"
650 PRINT@32,10;"";
660 A=5:B$="":GOSUB3160
670 IF ER=1 THEN GOTO  630
680 EC=VAL(B$)
690 IF EC<0 THEN GOTO  630
700 IF EC=0 THEN GOTO  830
710 GOSUB 3370
720 IF LI=0 THEN GOTO  760
730 PRINT@8,14;CHR$(27);"LENREGISTREMEN
T OCCUPE"
740 WAIT200
750 GOTO 610
760 ?WRITE1,EC'NO$,PN$,NR%,AD$,CP,VI$,T
E$
770 GOTO 600
780 '
790 GOSUB  3500
800 RETURN
810 '
820 '
830 ?WHERE1
840 N=PEEK(#48F)
850 FOR I=1 TO N
860 ?TAKE1,I'A$
870 IF A$<>CHR$(2) THEN GOTO  890
880 EC=I:I=N+1
890 NEXT I
900 ?WRITE1,EC'NO$,PN$,NR%,AD$,CP,VI$,T
E$
910 GOTO 600
920 '
930 '*********************
940 '* EFFACER UNE FICHE *
950 '*********************
960 '
```

```
970 GOSUB  3600
980 IF PB=1 THEN RETURN
990 CLS
1000 PRINT@7,10;"NUMERO D'ENREGISTREMEN
T: -----"
1010 PRINT@11,12;"0 = RETOUR AU MENU"
1020 PRINT@32,10;"";
1030 A=5:B$="":GOSUB 3160
1040 IF ER=1 THEN GOTO 1000
1050 EC=VAL(B$)
1060 IF EC<0 THEN GOTO1000
1070 IF EC=0 THEN GOTO 1210
1080 ?WHERE1
1090 NE=PEEK(#48F)
1100 IF EC>NE THEN GOTO 1000
1110 EF$=CHR$(2)
1120 GOSUB 3890
1130 CLS
1140 PRINT@9,14;"CONFIRMATION (O/N) ? -
"
1150 PRINT@30,14;"";
1160 GET A$
1170 PRINTA$;
1180 IF A$="O" THEN ?WRITE1,EC'EF$,EF$
1190 IF A$="N" THEN GOTO  990
1200 IF A$<>"O" THEN GOTO 1140 ELSE GOT
O  990
1210 GOSUB  3500
1220 RETURN
1230 '
1240 '*************************
1250 '* LIRE FICHE AVEC NUMERO *
1260 '*************************
1270 '
1280 CLS
1290 PRINT@4,13;"NOM DU FICHIER A OUVRI
```

```
R: ---------"
 1300 PRINT@29,13;"";
 1310 GOSUB  3630
 1320 IF PB=1 THEN RETURN
 1330 CLS
 1340 PRINT@7,10;"NUMERO D'ENREGISTREMEN
T: -----"
 1350 PRINT@11,12;"0 = RETOUR AU MENU"
 1360 PRINT@32,10;"";
 1370 A=5:B$="":GOSUB 3160
 1380 IF ER=1 THEN GOTO 1340
 1390 EC=VAL(B$)
 1400 IF EC<0 THEN GOTO 1340
 1410 IF EC=0 THEN GOTO 1480
 1420 !WHERE1
 1430 IF EC>PEEK(#48F) THEN GOTO 1340
 1440 GOSUB  3890
 1450 GOTO 1330
 1460 '
 1470 '
 1480 GOSUB  3500
 1490 RETURN
 1500 '
 1510 '**********************
 1520 '* LIRE FICHE AVEC NOM *
 1530 '**********************
 1540 '
 1550 CLS
 1560 PRINT@4,13;"NOM DU FICHIER A OUVRI
R: --------"
 1570 PRINT@29,13;"";
 1580 GOSUB  3630
 1590 IF PB=1 THEN RETURN
 1600 CLS
 1610 PRINT@12,14;"NOM: ----------"
 1620 PRINT@17,14;"";
```

```
1630 A=10:B$="":GOSUB 3160
1640 IF ER=1 THEN GOTO 1610
1650 IF B$="" THEN GOTO 1710
1660 CL$=B$
1670 GOSUB  4300
1680 GOTO 1600
1690 '
1700 '
1710 GOSUB  3500
1720 RETURN
1730 '
1740 '*************************
1750 '* LISTE FICHE PAR PRENOM *
1760 '*************************
1770 '
1780 CLS
1790 PRINT@4,13;"NOM DU FICHIER A OUVRI
R: --------"
1800 PRINT@29,13;"";
1810 GOSUB  3630
1820 IF PB=1 THEN RETURN
1830 CLS
1840 PRINT@12,14;"PRENOM: ----------"
1850 PRINT@20,14;"";
1860 A=10:B$="":GOSUB 3160
1870 IF ER=1 THEN GOTO 1840
1880 IF B$="" THEN GOTO 1940
1890 CL$=B$
1900 GOSUB  4460
1910 GOTO 1830
1920 '
1930 '
1940 GOSUB  3500
1950 RETURN
1960 '
1970 '****************************
```

```
1980 '* LISTE COMPLETE DU FICHIER *
1990 '*****************************
2000 '
2010 CLS
2020 PRINT@4,13;"NOM DU FICHIER A OUVRI
R: ---------"
2030 PRINT@29,13;"";
2040 GOSUB  3630
2050 IF PB=1 THEN RETURN
2060 ?WHERE1
2070 NE=PEEK(#48F)
2080 FOR EC=1TO NE
2090 ?TAKE1,EC'NO$
2100 IF NO$<>CHR$(2) THEN GOSUB  3890
2110 NEXT EC
2120 GOSUB 3500
2130 RETURN
2140 '
2150 '**********************
2160 '* SAISIE D'UNE FICHE *
2170 '**********************
2180 '
2190 CLS
2200 PRINTCHR$(17)
2210 PRINT@2,2;"NOM: ----------  PRENOM
 ----------"
2220 PRINT@10,6;"NUMERO DE RUE: -----"
2230 PRINT@10,10;"RUE: ----------------
---------"
2240 PRINT@15,12;"--------------------
----"
2250 PRINT@12,16;"CODE POSTAL: -----"
2260 PRINT@6,20;"VILLE: ---------------
-----"
2270 PRINT@3,24;"NUMERO DE TELEPHONE: -
                "
```

```
2280 PRINT CHR$(17):PRINT@7,2;"";
2290 A=10:B$="":GOSUB 3160
2300 PRINT CHR$(17)
2310 IF ER=0 THEN GOTO 2350
2320 PRINT@2,2;"NOM: ----------"
2330 PING
2340 GOTO 2280
2350 NO$=B$
2360 PRINTCHR$(17)
2370 IF NO$="" THEN RETURN
2380 PRINTCHR$(17)
2390 '
2400 PRINT CHR$(17): PRINT@27,2;"";
2410 A=10:B$="":GOSUB 3160
2420 PRINT CHR$(17)
2430 IF ER=0 THEN GOTO 2470
2440 PRINT@19,2;"PRENOM: ----------"
2450 PING
2460 GOTO 2400
2470 PN$=B$
2480 '
2490 PRINT CHR$(17):PRINT@25,6;"";
2500 A=5:B$="":GOSUB 3160
2510 PRINT CHR$(17)
2520 IF ER=0 THEN GOTO 2560
2530 PRINT@10,6;"NUMERO DE RUE: -----
2540 PING
2550 GOTO 2490
2560 NR%=VAL(B$)
2570 IF NR%<=0 THEN GOTO 2530
2580 '
2590 PRINT CHR$(17):PRINT@15,10;"";
2600 A=24:B$="":GOSUB 3160
2610 IF A$=CHR$(13) THEN GOTO 2700
2620 PRINTA$;:B$=B$+A$
2630 PRINT CHR$(17)
```

```
2640 PRINT CHR$(17):PRINT@15,12;"";:A=5
0:GOSUB 3160
2650 IF ER=0 THEN GOTO 2700
2660 PRINTCHR$(17):PRINT@10,10;"RUE: --
----------------------"
2670 PRINT@15,12;"----------------------
----"
2680 PING
2690 GOTO 2590
2700 AD$=B$:PRINT CHR$(17)
2710 '
2720 PRINTCHR$(17):PRINT@25,16;"";
2730 A=5:B$="":GOSUB 3160
2740 PRINTCHR$(17)
2750 IF ER=0 THEN GOTO 2800
2760 PRINT
2770 PRINT@12,16;"CODE POSTAL: -----"
2780 PING
2790 GOTO 2720
2800 CP=VAL(B$)
2810 IF CP<1 OR LEN(B$)<>5 THEN PING:GO
TO 2770
2820 '
2830 PRINT CHR$(17):PRINT@13,20;"";
2840 A=20:B$="":GOSUB 3160
2850 PRINT CHR$(17)
2860 IF ER=0 THEN GOTO 2910
2870 PRINT
2880 PRINT@6,20;"VILLE: ---------------
-----"
2890 PING
2900 GOTO 2830
2910 VI$=B$
2920 '
2930 PRINTCHR$(17):PRINT@24,24;"";:TE$=""
2940 FOR I=1 TO 5
```

```
2950 A=1:B$="":GOSUB 3160
2960 PRINTA$;" ";
2970 TE$=TE$+B$+A$+" "
2980 NEXT I
2990 PRINTCHR$(17)
3000 IF LEN(TE$)=15 THEN GOTO 3050
3010 PRINT
3020 PRINT@3,24;"NUMERO DE TELEPHONE: -
- -- -- -- --"
3030 PING:TE$=""
3040 GOTO 2930
3050 PRINT CHR$(17):PRINT@15,25;"FICHE
OK..? -";:PRINT CHR$(8);
3060 GET A$
3070 PRINT@27,25;A$;
3080 IF A$="N" THEN GOTO 2190
3090 IF A$="O" THEN RETURN
3100 PRINT CHR$(8);:GOTO3060
3110 '
3120 '************************
3130 '* SAISIE DE CARACTERES *
3140 '************************
3150 '
3160 GET A$
3170 IF A$=CHR$(16) THEN GOTO 3290
3180 IF LEN(B$)=A AND A$<>CHR$(13) AND
A$<>CHR$(127) THEN GOTO 3250
3190 IF A$=CHR$(13) THEN GOTO 3270
3200 IF A$<>CHR$(127) THEN GOTO 3230
3210 IF LEN(B$)>0 THEN B$=LEFT$(B$,LEN(
B$)-1):PRINT CHR$(8);"-";CHR$(8);
3220 GOTO 3160
3230 B$=B$+A$:PRINT A$;
3240 GOTO 3160
3250 ER=1
3260 RETURN
```

```
3270 ER=0
3280 RETURN
3290 IF PEEK(48020)=32 THEN IM=1 ELSE I
M=0
3300 IF PEEK(48020)=32 THEN POKE 48020,
73 ELSE POKE 48020,32
3310 GOTO3160
3320 '
3330 '*************************
3340 '* VERIFICATION ENR. LIBRE *
3350 '*************************
3360 '
3370 ?WHERE 1
3380 N=PEEK(#48F)
3390 IF EC>N+1 THEN GOTO  3430
3400 ?TAKE1,EC'A$
3410 IF A$<>CHR$(2) THEN LI=1 ELSE LI=0
3420 RETURN
3430 LI=1
3440 RETURN
3450 '
3460 '************************
3470 '* FERMETURE DU FICHIER *
3480 '************************
3490 '
3500 ?CLOSE1
3510 FOR I=1 TO 8
3520 POKE48000+I,32
3530 NEXT I
3540 RETURN
3550 '
3560 '************************
3570 '* OUVERTURE DU FICHIER *
3580 '************************
3590 '
3600 CLS
```

```
3610 PRINT@3,13;"NOM DU FICHIER A MODIF
IER: --------"
3620 PRINT@30,13;"";
3630 A=8:B$="":GOSUB 3160
3640 IF ER=1 THEN GOTO 3610
3650 IF B$="" THEN PB=1:RETURN
3660 N$=B$+"=1"
3670 ?ERSET
3680 ?OPEN N$
3690 ?ERR GOTO 3760
3700 ?EROFF
3710 FORI=1TOLEN(B$)
3720 POKE 48000+I,ASC(MID$(B$,I,1))
3730 NEXT I
3740 PB=0
3750 RETURN
3760 IF PEEK(#489)<>7 THEN GOTO 3810
3770 CLS
3780 PRINT@11,14;"FICHIER INEXISTANT";
3790 WAIT200
3800 PB=1:RETURN
3810 GOSUB 3500
3820 PB=1
3830 RETURN
3840 '
3850 '************************
3860 '* EDITION D'UNE FICHE *
3870 '************************
3880 '
3890 CLS
3900 ?ERSET
3910 ?TAKE1,EC'NO$,PN$,NR%,AD$,CP,VI$,T
E$
3920 ?ERR GOTO 4200
3930 ?EROFF
3940 PRINT@35,0;EC
```

```
3950 PRINT@2,2;"NOM: ";NO$
3960 PRINT@19,2;"PRENOM: ";PN$
3970 PRINT@10,6;"NUMERO DE RUE: ";NR%
3980 IF LEN(AD$)<=25 THEN AH$=AD$:AB$="
"
3990 IF LEN(AD$)>25 THEN AH$=LEFT$(AD$,
25):AB$=MID$(AD$,26)
4000 PRINT@10,10;"RUE: ";AH$
4010 PRINT@15,12;AB$
4020 PRINT@12,16;"CODE POSTAL: ";CP
4030 PRINT@6,20;"VILLE: ";VI$
4040 PRINT@3,24;"NUMERO DE TELEPHONE: "
;TE$
4050 WAIT 200
4060 IF KEY$="" THEN GOTO  4090
4070 PRINT@3,25;"PRESSEZ UNE TOUCHE POU
R CONTINUER"
4080 GET A$
4090 IF IM=0 THEN GOTO  4190
4100 LPRINT TAB(2);"NOM: ";NO$;
4110 LPRINT TAB(19);"PRENOM: ";PN$
4120 LPRINT TAB(10);"NUMERO DE RUE: ";N
R%
4130 LPRINT TAB(10);"RUE: ";AH$
4140 LPRINT TAB(15);AB$
4150 LPRINT TAB(12);"CODE POSTAL: ";CP
4160 LPRINT TAB(6);"VILLE: ";VI$
4170 LPRINT TAB(3);"NUMERO DE TELEPHONE
: ";TE$
4180 LPRINT:LPRINT
4190 RETURN
4200 IF PEEK(#489)<>16 THEN STOP
4210 CLS
4220 PRINT@10,14;"ENREGISTREMENT LIBRE"

4230 WAIT 200
```

```
4240 RETURN
4250 '
4260 '*****************************
4270 '* LECTURE DU FICHIER PAR CLE *
4280 '*****************************
4290 '
4300 ?WHERE1
4310 N=PEEK(#48F)
4320 FOR I=1 TO N
4330 ?TAKE1,I'A$
4340 IF A$=CHR$(2) THEN GOTO  4380
4350 EC=I
4360 IF A$=CLE$ THEN GOSUB  3890 ELSE G
OTO  4380
4370 FL=1
4380 NEXT I
4390 CLS
4400 IF FL=1 GOTO  4440
4410 PRINT@6,14;"LE NOM ";CL$;" N'EXIST
E PAS"
4420 PRINT@10,16;"DANS CE FICHIER"
4430 WAIT 500
4440 FL=0
4450 RETURN
4460 ?WHERE1
4470 N=PEEK(#48F)
4480 FOR I=1 TO N
4490 ?TAKE1,I'NO$,PN$
4500 IF NO$=CHR$(2) THEN GOTO  4540
4510 EC=I
4520 IF PN$=CLE$ THEN GOSUB  3890 ELSE
GOTO  4540
4530 FL=1
4540 NEXT I
4550 CLS
4560 IF FL=1 GOTO  4600
```

```
4570 PRINT@6,14;"LE PRENOM ";CL$;" N'EX
ISTE PAS"
4580 PRINT@10,16;"DANS CE FICHIER"
4590 WAIT 500
4600 FL=0
4610 RETURN
```

Exemple d'exécution : Edition d'une fiche.

```
NOM: DUPONT        PRENOM: MICHEL
        NUMERO DE RUE:  43
        RUE: AVENUE DES PLATANES
             RESIDENCE BELLE VUE
          CODE POSTAL:  13850
    VILLE: MARSEILLE
 NUMERO DE TELEPHONE: 16 91 23 56 48


NOM: RACINE        PRENOM: ALFRED
        NUMERO DE RUE:  581
        RUE: BOULEVARD ST GERMAIN

          CODE POSTAL:  75034
    VILLE: PARIS
 NUMERO DE TELEPHONE: 16 17 32 45 32
```

Commentaires :

LIGNES 10-70 : Initialisation des variables alphanumériques utilisées pour l'accès aux informations du fichier.

Demande de copie sur imprimante : chaque édition de fiche à l'écran sera répercutée sur l'imprimante.

LIGNES 100-330 : affichage du menu, attente de la réponse de l'utilisateur : si elle correspond à une option, celle-ci est exécutée, sinon le menu est reproposé avec obstination.

```
                                        CAPS

NOM DU FICHIER ? TELEFON
```

```
                                        CAPS
CREER UN FICHIER = CRE
INSERER UNE FICHE = INS
EFFACER UNE FICHE = EFF
LIRE FICHE A PARTIR D'UN NUMERO = LNU
LIRE FICHE A PARTIR D'UN NOM = LNO
LIRE FICHE A PARTIR D'UN PRENOM = LPR
EDITION TOTALE DU FICHIER =EDI
FIN = FIN
```

```
TELEFON                                  CAPS

 NOM: AZERTY----  PRENOM: PIERRE----

        NUMERO DE RUE: 16---

        RUE: MARECHAL JUIN-----------
             ------------------------

          CODE POSTAL: 13245

    VILLE: MARSEILLE-----------

 NUMERO DE TELEPHONE: 16 24 35 67 4
```

```
TELEFON                                  CAPS


      NUMERO D'ENREGISTREMENT: 567


        0 = LE PREMIER LIBRE
```

LIGNES 350-520 : Option création d'un fichier.

On peut ainsi créer plusieurs fichiers d'adresses (adresses personnelles, adresses professionnelles, etc...).

Si aucun nom de fichier n'est donné au moment de l'appui sur RETURN on retourne au menu.

Dès que le fichier est créé, il est initialisé ; on écrit séquentiellement le code ASCII 2 dans la 1re zone de chacun des enregistrements.

LIGNES 550-920 : option création d'une nouvelle fiche.

Il est d'abord demandé sur quel fichier on veut travailler ce fichier est alors ouvert. Ensuite un masque de saisie est affiché à l'écran, il permet de remplir commodément la fiche. Ensuite, deux possibilités sont offertes donner un numéro d'enregistrement ou placer la fiche dans le premier enregistrement libre. Dans le premier cas avant d'enregistrer la fiche, on vérifie que cet emplacement est libre.

LIGNES 940-1220 : Effacement d'une fiche.

Les préliminaires sont identiques à ceux de l'option précédante.

Ensuite, le numéro de la fiche à effacer est demandé. Comme c'est une opération importante, le contenu de la fiche est présenté sur l'écran (pendant 5 sec, mais ce temps peut être prolongé en appuyant sur une touche) ensuite une confirmation est demandée avant que la fiche ne soit effacée définitivement.

LIGNES 1250-1500 : Option lecture d'une fiche d'après son numéro d'enregistrement.

Le nom du fichier où prendre la fiche est demandé, puis le numéro de la fiche à éditer. Si le numéro donné est Ø on retourne au menu. Des contrôles sont effectués par sécurité (numéros de fiche négatifs où à l'extérieur du domaine du fichier).

LIGNES 1520-1750 : option lecture d'une fiche avec un nom comme clé.

Après l'ouverture de rigueur, c'est évidemment un nom qui est demandé. Ensuite la totalité du fichier est lue, chaque fois que NO$ et CL$ correspondent la fiche est éditée. Si le nom n'existe pas un message d'erreur est produit, puis un autre nom est demandé. Un nom vide renvoie au menu.

LIGNES 1750-1960 : Option lecture par prénom. Le principe est similaire à l'option LNO ; c'est un prénom qui est demandé.

LIGNES 1970-2130 : Option liste complète du fichier.

Toutes les fiches non vides sont éditées grâce à cette option. Elle permet de faire périodiquement le point sur l'état du fichier.

LIGNES 2160-3130 : Masque de saisie.

Dans la pratique, presque tous les programmes interactifs utilisant des fichiers ont recours au masque de saisie, car il libère l'utilisateur des contraintes de longueur des enregistrements en lui signalant toute erreur ; il permet une bonne présentation par sa structure plein écran. Pour un programme d'utilisation professionnelle il ne faut pas hésiter à le soigner, c'est lui qui rend le travail agréable et sûr.

Le masque défini pour ce programme contrôle la longueur de chaque constituant d'un enregistrement. Si un élément devient trop long il est totalement effacé, et son introduction est redemandée. On contrôle que le numéro de rue soit bien numérique, que le code postal sait un nombre à 5 chiffres. Toute erreur est signalée par un PING de façon à bien attirer l'attention.

LIGNES 3130-3320 : Saisie de caractères.

Cette routine attend l'entrée de caractères au clavier.

Si les touches CTRL et P sont appuyées simultanément on change le mode de l'imprimante si elle est ON, elle devient OFF et vice-versa.

Si c'est la touche DEL qui est appuyée le dernier caractère est effacé. RETURN ou un mot trop long provoque le retour au programme appelant.

LIGNES 3340-3450 : Vérification enregistrement libre.

On vérifie que le premier champ de l'enregistrement EC contient bien le caractère de code 2. LI vaut alors Ø, sinon il vaut 1.

LIGNES 3470-3550 : Fermeture du fichier.

Le fichier est refermé, son nom est effacé de la ligne des status.

LIGNES 3570-3860 : Ouverture du fichier.

Le nom du fichier est demandé, ensuite on contrôle si il existe bien sur la disquette. Dans ce cas, on écrit son nom sur la même ligne que CAPS (lignes des STATUS).

LIGNES 3860-4250 : Edition d'une fiche.

Lecture de l'enregistrement de numéro EC, si cet enregistrement est libre la lecture provoque une ERREUR OUT OF DATA on contrôle ce cas à l'aide des instructions ! ERSET et ! ERR GOTO. Ensuite, la fiche est présentée sur l'écran pendant 2 sec. si on ne touche pas au clavier, sinon on attend l'appui sur une autre touche pour continuer. Si l'indicateur d'imprimante est armé la fiche est aussi éditée sur l'imprimante.

LIGNES 3340-3450 : Vérification enregistrement.

Le fichier est lu en entier, à la recherche de la clé CL$, si elle existe, l'enregistrement complet est édité, sinon on affiche le message.

{ LE NOM
{ PRENOM « TOTO » N'EXISTE PAS DANS CE FICHIER.

TRI

Nous allons voir quelques algorithmes de tri parmi les plus simples car ils sont très employés pour les fichiers. Classements par ordre alphabétique d'un fichier client ou d'un fichier d'adresse. Classement par ordre numérique d'un fichier de stock de pièces (ou des bouteilles de votre cave)...

1° - TRI RIPPLE (RIPPLE SORT)

Le principe consiste à comparer 2 éléments adjacents et à les inverser si le premier est supérieur au second. L'algorithme peut être le suivant :

```
Début : Tri ripple
indicateur = 1
tant que indicateur = 1
    faire I = 1 : Indicateur = 0
    faire I = S
    répéter jusqu'à la fin de la table
        comparer élément I et élément I + 1
        si élément I › élément I + 1 faire
                                    Inverser élément I et élément I + 1
                                    Indicateur = 1
        finsi                       fait
    fait
fin tri ripple
```

RIPPLE

```
5 ' **************
10 '* TRI RIPPLE *
20 '**************
30 I=N-1
40 REPEAT
45 INDIC=FALSE
50 FOR J=1TOI
60 IF A(J)<=A(J+1) THEN GOTO 90
70 X=A(J):A(J)=A(J+1):A(J+1)=X
80 INDIC=TRUE
90 NEXT J
95 I=I-1
100 UNTIL INDIC=FALSE
```

Ce programme permet de classer des nombres rangés au préalable dans A (), il faut aussi spécifier la valeur N = nombre d'éléments à classer.

2° - TRI A BULLE (BUBBLE SORT)

On compare le premier élément à tous les successeurs et on inverse les deux éléments si le second est plus petit que le premier. Ainsi à la fin de la première passe le plus petit élément est au sommet (il est remonté comme une bulle d'où le nom) on recommence pour le deuxième éléments... jusqu'au dernier.

Algorithme :

Début : Tri à bulle
pour I = 1 **à** nombre d'éléments - 1
faire pour J = I + 1 **à** nombre d'éléments
    faire comparer A (I) et A (J)
        si A (I) › A (J) **alors** inverser A (I) et A (J)
        finsi
        J = J + 1
    fait
I = I + 1
fait
fin tri à bulle

BUBBLE

```
10 '***************
20 '* TRI A BULLE *
30 '***************
40 '
50 FOR I=1 TO N-1
60 FOR J=I+1 TO N
70 IF A(I)<=A(J) THEN GOTO 100
90 X=A(I):A(I)=A(J):A(J)=X
100 NEXT J
110 NEXT I
120 '
130 END
```

Les conditions d'exécution sont les mêmes que pour le tri RIPPLE.

Ces deux méthodes de tri sont les plus simples mais aussi les plus lentes car il faut environ n^2 comparaisons pour classer une table (ceci dépend de la distribution des éléments) ; néanmoins pour des tables de moins de 50 éléments ces deux tris ont un bon rapport complexité / temps d'exécution. Le premier est aussi intéressant pour des tables presque en ordre puisque il peut être arrêté par le test de l'indicateur. Les tris qui vont suivre sont plus performants et plus complexes.

3° - TRI SHELL

Le principe est le même que pour le tri RIPPLE, mais la méthode est accélérée par des pré-classements successifs plus rapides.
Exemple d'exécution (voir listing programme).

Soit à classer :

↓		**Tri ripple**				
10	8	7	6	5	1	1
8	7	6	5	1	5	5
7	6	5	1	6	6	6
6	5	1	7	7	7	7
5	1	8	8	8	8	8
1	10	10	10	10	10	10
Indic	1	1	1	1	1	0

Pour classer la liste il faut 6 tours de boucles qui comprennent chacun 5 comparaisons. 36 comparaisons au total.

Tri shell

au début Pas = 3 J = I + 1 ; ensuite Pas = 1 J = I + 1 tri ripple normal

10	6	6	5	1	1
8	5	5	1	5	5
7	1	1	6	6	6
6	10	10	8	7	7
5	8	8	7	8	8
1	7	7	10	10	10
Indice	1	0	1	1	0

Il ne faut que 5 tours de boucles pour classer la liste. Sur les 5 tours 2 comprennent 3 comparaisons, 3 comprennent 5 comparaisons. 21 comparaisons.

Cette méthode est plus rapide car les petits éléments placés en fin de liste remontent plus vite qu'avec le tri RIPPLE ou à Bulle.

SHELL

```
10 '*********************
20 '* TRI METHODE SHELL *
30 '*********************
40 '
50 PAS=INT(N/2)
60 REPEAT
70 INDIC=FALSE
80 FOR I= 1 TO N-PAS
90 J=I+PAS
100 IF A(I)<=A(J) THEN GOTO 130
110 X=A(I):A(I)=A(J):A(J)=X
120 INDIC=TRUE
130 NEXT I
140 IF INDIC=TRUE THEN GOTO 70
150 PAS=INT(PAS/2)
160 UNTIL PAS=0
170 '
```

2° - TRI SHELL - METZNER

Il s'agit d'une amélioration de rapidité du tri-shell. La méthode qui offre le meilleur rapport complexité/ efficacité pour classer la liste précédante cet algorithme requiert seulement 12 comparaisons.

Son principe : il débute comme un tri shell, mais chaque fois que l'on permute deux éléments on regarde aussi si il faut le permuter avec les précédants ainsi on obtient un classement en peu de comparaisons.

Exemple des permutations successives pour la liste

10	6	6	6	5	5	1	1	1	1	1	1	1
8	8	5	5	6	1	5	5	5	5	5	5	5
7	7	7	1	1	6	6	6	6	6	6	6	6
6	10	10	10	10	10	10	10	8	8	8	7	7
5	5	8	8	8	8	8	8	10	10	7	8	8
1	1	1	7	7	7	7	7	7	7	0	10	10

Chaque colonne ne représente plus comme précédemment un tour de boucle mais une comparaison.

```
SHMETZNE

10 '*****************************
20 '* TRI METHODE SHELL-METZNER *
30 '*****************************
40 '
50 PAS=INT(N/2)
60 REPEAT
70 J=1
80 K=N-PAS
90 L=J
95 REPEAT
100 M=L+PAS
110 IF A(L)<=A(M) THEN L=0:GOTO 140
120 X=A(L):A(L)=A(M):A(M)=X
130 L=L-PAS
140 UNTIL L<1
150 J=J+1
```

```
160 IF J <=K THEN GOTO 90
170 PAS=INT(PAS/2)
180 UNTIL PAS=0
190 '
```

5° - TRI RAPIDE : QUICK SORT

Il s'agit d'une méthode de tri assez évoluée mais très rapide (comme son nom l'indique). Il en existe plusieurs variantes plus ou moins sophistiquées ; nous avons choisi un algorithme performant avec choix du pivot. Cette méthode est assez lourde à mettre en œuvre et ne se justifie que pour des tris importants : + de 100 éléments.

Commentaires :
A () : table des nombres à trier
PLE (,) : table pile (pile à 2 dimensions)
PP : pointeur de pile ; il pointe vers les éléments du sommet de la pile.
LP : longueur d'une partition, voir explication plus loin
BAS : il pointe vers l'élément A (1) au début, puis sur le premier élément de la partition à trier.
HAUT : il pointe vers l'élément A (256)* au début, puis sur le dernier élément de la partition à trier *(pour une table de 256 éléments)
PVOT : contient la valeur du pivot pour effectuer les comparaisons.
MIL : il pointe vers le milieu de la partition à trier, au début milieu de la table
N : nombre d'éléments à classer
PB, PH : pointeur bas, pointeur haut ; ils permettent le classement des éléments autour du pivot.

Le programme de tri commence réellement en ligne 110 ; les lignes 7-8 sont le sous-programme d'inversion de deux éléments il est placé en début de programme car l'accès y est ainsi plus rapide.

10-50 : Ces instructions permettent de remplir une table avec des nombres aléatoires compris entre 0 et 999 ; ce sont les nombres qui vont être classés.

110-150 : choix du pivot. Le principe du tri consiste à répartir de part et d'autre d'un élément nommé pivot, tous les autres de telle sorte que les éléments plus petits soient sous le pivot, tandis que les plus grands seront au dessus.

QUICKSOR

```
0 GOTO 10
1 '*********************************
2 '* INITIALISATION TABLE NOMBRES *
3 '*********************************
4 '
7 X=A(U):A(U)=A(S):A(S)=X
8 RETURN
9 '
10 DIM A(256)
20 FOR I=1 TO 256
30 A(I)=INT(RND(1)*1000)
40 NEXT I
45 PRINT"DEBUT TRI"
50 '
51 '****************************
52 '* INITIALISATION VARIABLES *
53 '****************************
54 '
60 DIM PLE(1,14)
70 PP=0:LP=4:BAS=1:HAUT=256
80 PVOT=0:MIL=0:N=256
90 '
99 ' ******************
100 '* CHOIX DU PIVOT *
101 '******************
102 '
110 MIL=INT(BAS+HAUT)/2
120 IF A(BAS)<A(MIL) THEN U=BAS:S=MILT:
GOSUB 7:I=0
130 IF A(HAUT)<A(BAS) THEN U=BAS:S=HAUT
:GOSUB 7:I=1
140 IF I=1 AND A(BAS)<A(MIL) THEN U=MIL
:S=BAS:GOSUB 7
150 '
```

```
151 '***************************
152 '* CLASSEMENT DES ELEMENTS *
153 '*      AUTOUR DU PIVOT     *
154 '***************************
160 PVOT=A(BAS)
170 PB=BAS:PH=HAUT+1
180 REPEAT
190 PB=PB+1
200 UNTIL A(PB)>=PVOT
210 REPEAT
220 PH=PH-1
230 UNTIL A(PH)<=PVOT
240 IF PB>=PH THEN GOTO 270
250 U=PB:S=PH:GOSUB 7
260 GOTO 180
270 U=BAS:S=PH:GOSUB 7
280 '
281 '***************************
282 '* ORGANISATION DE LA PILE *
283 '***************************
284 '
290 IF (PH-BAS)>(HAUT-PH) THEN GOTO 400
298 '
300 IF (PH-BAS)<=LP THEN GOTO 360
310 PP=PP+1
320 PLE(0,PP)=PH+1
330 PLE(1,PP)=HAUT
340 HAUT=PH-1
350 GOTO 110
360 IF (HAUT-PH)<= LP THEN GOTO 500
370 BAS=PH+1
380 GOTO 110
395 '
400 IF(HAUT-PH)<=LP THEN GOTO 460
410 PP=PP+1
```

```
420 PLE(0,PP)=BAS
430 PLE(1,PP)=PH-1
440 BAS=PH+1
450 GOTO 110
460 IF (PH-BAS)<=LP THEN GOTO 500
470 HAUT=PH-1
480 GOTO 110
490 '
491 '***********
492 '* DEPILER *
493 '***********
494 '
500 IF PP=0 THEN GOTO 600
510 BAS=PLE(0,PP)
520 HAUT=PLE(1,PP)
530 PP=PP-1
540 GOTO 110
587 '
589 '********************
590 '* TRI FINAL RIPPLE *
591 '********************
592 '
600 I=N-1
601 PRINT"DEBUT RIPPLE"
620 REPEAT
630 INDIC=FALSE
640 FOR J=1 TO I
650 IF A(J)<=A(J+1) THEN GOTO 680
660 X=A(J):A(J)=A(J+1):A(J+1)=X
670 INDIC=TRUE
680 NEXT J
685 I=I-1
690 UNTIL INDIC=FALSE
691 '
```

```
692 '************************
693 '* EDITION DES RESULTATS *
694 '************************
695 '
700 FOR I=1 TO N STEP 5
710 PRINTA(I),A(I+1),A(I+2),A(I+3),A(I+
4)
720 NEXT I
730 END
```

Pour un maximum d'efficacité, il faut que le pivot ait une valeur moyenne de telle sorte qu'on ait autant d'éléments au-dessus que au-dessous. Ici, le pivot est choisi parmi trois éléments : le premier de la partition, le dernier et l'élément milieu. Celui qui a la valeur intermédiaire est placé en début de table.

160-270 : classement des éléments autour du pivot ; ceux qui sont plus petits que le pivot au-dessous, ceux qui sont plus grands au-dessus.

290-490 : quand le tri précédant est terminé on se retrouve avec deux partitions : les éléments plus petits que le pivot et les éléments plus grands que le pivot. Dans chacune des partitions on choisit un nouveau pivot et on effectue de nouveau le traitement.

Mais comme on ne peut traiter simultanément les 2 partitions il faut en choisir une et mémoriser les coordonnées de l'autre. On choisit de classer en premier la partition qui contient le moins d'éléments et de conserver les coordonnées de l'autre dans une pile :
PLE (O,X) = Adresse du 1er élément de la partition
PLE (1,X) = Adresse du dernier élément de la partition.

Chaque traitement autour du pivot conduit à deux nouvelles partitions. On peut théoriquement aboutir à des partitions qui ne contiendraient qu'un seul élément ; mais on démontre alors que le tri perd son efficacité. Donc on décide que la partition sera considérée comme classée dès qu'elle aura une longueur inférieure ou égale à LP = 4. On terminera le tri avec une autre méthode : Ripple est bien adapté (table partiellement classée).

500-540 : quand une partition contient moins de 5 éléments, on cherche les coordonnées de celles précédemment mémorisées.

600-690 : quand la pile est vide, il n'y a plus de partition à traiter, on finit le tri par la méthode tri-RIPPLE.

700-730 : Edition des résultats (attention dans ce programme elle termine sur une erreur car on essaie de faire afficher A (257), A (258), A (259), A (260)]

Exemple d'exécution commenté :

Soit à classer :

7,9,16,5,9,14,2,12,8,1,10,15,4,6,11,3

Choix du pivot : BAS = 1 HAUT = 16 PVOT = 0 MIL = 8

120 : A (1) ‹ A (8) OUI ⧫	12,9,... ...2,7,8,... ...11,3
130 : A (16) ‹ A (1) OUI ⧫	3,9,... ...2,7,8,... ...11,12
140 : I = 1 et A (1) ‹ A (7) ⧫	7,9,... ...2,3,8,... ...11,12
160 : PVOT = 7 PBAS = 1 PH = 17	
180,190,200 : PB = 2	A (2) › PVOT
210,220,230 : PH = 16	A (16) › PVOT
210,220,230 : PH = 15	A (15) › PVOT
210,220,230 : PH = 14	A (14)<PVOT
240,250,260 : ⧫ A (14) = A (2) et A (2) = A (14)	

[7,] 6,16,5,9,14,2,3,8,1,10,15,4,9,11,12

180 190 200 : PB = 3	A (3) › PVOT
210 220 230 : PH = 13	A (13) ‹ PVOT
240-250-260 : ⧫ A (13)⋚A (3)	

[7,] 6,4,5,9,14,2,3,8,1,10,15,16,9,11,12

180 190 200 : PB = 4	A (4) ‹ PVOT
180 190 200 : PB = 5	A (5) › PVOT
210 220 230 : PH = 12	A (12) › PVOT
210 220 230 : PH = 11	A (11) › PVOT
210 220 230 : PH = 10	A (10) ‹ PVOT
240-250-260 : A (10)⋚A (5)	

⋚ : inverser

[7,] 6,4,5,1,14,2,3,8,9,10,15,16,9,11,12

180,190,200 : PB = 6	A (6) › PVOT
210,220,230 : PH = 9	A (9) › PVOT
210,220,230 : PH = 8	A (8) ‹ PVOT
240-250-260 : A (8)⋚A (6)	

[7,] 6,4,5,1,3,2,14,8,9,10,15,16,9,11,12

180 190 200 : PB = 7	A (7) ‹ PVOT
180 190 200 : PB = 8	A (8) › PVOT
210 220 230 : PH = 7	A (7) ‹ PVOT
240 PB>PH : A (1)⋚A (7)	

2,6,4,5,1,3,[7,] 14,8,9,10,15,16,9,11,12

PH-BAS = 7-1 = 6	HAUT-PH = 16-7 = 9
290 ⧫ : 6 ‹ 9 ⧫ 300	
300 : 6 › LP (LP = 4) ⧫ 310 : PP = 1	PLE (0,1) = 8 PLE (1,1) = 16 HAUT = 6

Maintenant, on a mémorisé l'adresse début et de fin de la partition la plus longue : elle commence au 8e élément et finie au 16e élément. On retourne en 110 ou on recommence avec BAS = 1 HAUT = 6

CHOIX DU PIVOT : [3,] 6,4,5,1,4 [7,] 14,8,9,10,15,16,9,11,12

CLASSEMENT : 2,1,[3,] 5,6,4,[7,] 14,8,9,10,15,16,9,11,12

290 ⧫ 300 : PH-BAS = 2 ‹ LP = 4 ⧫ 360
290 ⧫ 360 HAUT-PH = 3 ‹ LP ⧫ 500
290 ⧫ 500 : PP = 1 ⧫ 510 BAS = 8 HAUT = 16 PP = Ø

On recommence mais on traite la 2e partition.

CHOIX DU PIVOT : 2,1,3,5,6,4,7, 14, 8,9,10,12,16,9,11,15

CLASSEMENT : 2,1,3,5,6,4,7,9,8,9,10,12,11, [14,] 16,15

290 : ⧫ PH-BAS = 6 HAUT-PH = 2 ⧫ 400
400 : 2 ‹ LP ⧫ 460
460 : 6 › LP HAUT = 13

CHOIX DU PIVOT : 2,1,3,5,6,4,7,[9,] 8,9,10,12,11,14,16,15

CLASSEMENT : 2,1,3,5,6,4,7, 9, 8,[9,] 10,12,11,14,16,15

290 : ⧫ PH-BAS = 2 HAUT-PH = 3 ⧫ 3ØØ
300 : PH-BAS ‹ LP ⧫ 360
360 : HAUT-PH ‹ LP ⧫ 500
500 : PP = 0 ⧫ 600

On exécute le tri final puisque toutes les partitions ont une longueur ≤ 4 : 2,1,[3],5,6,4,[7],9,8,[9],10,12,11,[14],16,15
le tri Ripple étant exécuté on affiche les résultats. Le classement est réalisé en 3 tours de boucle.

Pour un tableau de 256 nombres aléatoires entiers à 3 chiffres, il faut en moyenne par cette méthode 1 mn 20 s. dont environ 40 s. pour le Quick sort et 40 s. pour le Tri Ripple, alors que le tri Ripple seul demande environ 20 mn.

6° - TRIS ALPHANUMERIQUES :

Le BASIC permet de comparer des chaînes de caractères ; il sait discerner entre deux chaînes qu'elle est la plus « petite » (comprendre celle qui commence par les lettres les plus faibles dans l'alphabet). Par exemple : ,«ARN», «ADN», «ANR», et «CTN» seront classées comme suit : «ADN», «ANR», «ARN», «CTN».

Les algorithmes précédants restent entièrement valables pour les variables alphanumériques. (Des problèmes apparaîtront si minuscules et majuscules sont mélangées «W» est plus petit que «a»). A titre d'exemple les algorithmes RIPPLE et Bubble sont reproduits ci-après :

ARIPPLE

```
5 ' **************
10 '* TRI RIPPLE *
20 '**************
30 I=N-1
40 REPEAT
45 INDIC=FALSE
50 FOR J=1TOI
60 IF A$(J)<=A$(J+1) THEN GOTO 90
70 X=A$(J):A$(J)=A$(J+1):A$(J+1)=X
80 INDIC=TRUE
90 NEXT J
95 I=I-1
100 UNTIL INDIC=FALSE
```

ABUBBLE

```
10 '******************************
20 '* TRI A BULLE ALPHANUMERIQUE *
30 '******************************
40 '
50 FOR I=1 TO N-1
60 FOR J=I+1 TO N
70 IF A$(I)<=A$(J) THEN GOTO 100
90 X$=A$(I):A$(I)=A$(J):A$(J)=X$
100 NEXT J
110 NEXT I
120 '
130 END
```

FICHIERS A ACCES DIRECT
PROGRAMMATION AVANCEE

I - ACCES PAR CLE :

Nous avons vu que l'on pouvait accéder à un enregistrement que si l'on connaissait son numéro logique. Malheureusement dans la plupart des cas ce numéro est inconnu ou mal pratique, et c'est par une clé (NOM, ADRESSE...) que l'on veut accéder à l'enregistrement. La première solution (CF programme agenda) est de lire séquentiellement le fichier jusqu'à ce que la clé et l'enregistrement lu correspondent, mais alors on perd tous les avantages du fichier à accès direct. D'autres méthodes ont donc été développées.

1° - La plus simple est l'accès indexé avec constitution d'index en début de travail.

Au moment d'utiliser le fichier, on lit en séquence le champ clé et on forme une table d'index en mémoire centrale. Pour accéder à un enregistrement on cherche séquentiellement dans la table d'index jusqu'à ce que la clé et le contenu de la table d'index concordent, alors le rang donne le numéro d'enregistrement. Cette méthode est plus rapide car on effectue un seul accès au disque. Par contre, pour un fichier volumineux la constitution de la table d'index peut être pénalisante en temps.

Exemple
Table d'index :

LUC
JACQUES
PAUL
.
.
.
FRANÇOIS

CLE = « PAUL »
On lit séquentiellement la table
Paul est le 3e élément donc
l'enregistrement qui contient Paul est le 3e

On y accède par l'instruction ! TAKE1,3'N0$,...

2o - Cette méthode peut être améliorée en constituant 2 tables d'index : une pour la clé et une pour le numéro d'enregistrement correspondant. Ensuite, la table CLE est triée par ordre alphabétique (si CLE alpha). La recherche d'un élément peut dès lors s'effectuer par dichotomie. On converge beaucoup plus vite vers l'élément recherché.

Exemple :

Un fichier d'élèves est constitué par des noms et des prénoms. On veut accéder au prénom d'un élève en connaissant son nom. Les enregistrements ont été effectués au moment de l'inscription à l'école donc pas, en suivant un ordre quelconque.

1 MARTIN Jean
2 PETIT Marie
3 MOTA Alain
4 DECCA François
5 ORRA Jules
6 SENS Paul
7 CLIR Michel
8 DUPONT Pierre
9 PAPER Odile

On constitue les tables d'index N0$ et INDEX

N0$ (1) = MARTIN	INDEX (1) = 1
N0$ (2) = PETIT	INDEX (2) = 2
N0$ (3) = MOTA	INDEX (3) = 3
N0$ (4) = DECCA	INDEX (4) = 4
N0$ (5) = ORRA	INDEX (5) = 5
N0$ (6) = SENS	INDEX (6) = 6
N0$ (7) = CLIR	INDEX (7) = 7
N0$ (8) = DUPONT	INDEX (8) = 8
N0$ (9) = PAPER	INDEX (9) = 9

Ensuite la table N0$ est classée, mais la table INDEX doit suivre car INDEX (I) indique le numéro d'enregistrement au on trouve le nom N0$ (I).

Table classée :

N0$ (1) = CLIR	INDEX (1) = 7
N0$ (2) = DECCA	INDEX (2) = 4
N0$ (3) = DUPONT	INDEX (3) = 8
N0$ (4) = MARTIN	INDEX (4) = 1
N0$ (7) = PAPER	INDEX (7) = 9
N0$ (9) = SENS	INDEX (9) = 6

Maintenant pour trouver le numéro d'enregistrement correspondant PAPER on précède par dichotomie :

CLIR ◀ BAS
DECCA
DUPONT
MARTIN
MOTTA ◀ MIL
ORRA ◀ BAS'
PAPER ◀ MIL'
PETIT
SENS ◀ HAUT

On compare PAPER (la clé) avec l'élément milieu de la table Si CLE › N0$ (MIL) alors BAS = MIL + 1 sinon HAUT = MIL - 1, SI CLE = N0$ (MIL) on a trouvé l'élément cherché, si BAS › HAUT l'élément n'est pas dans la table : Ici PAPER › MOTTA donc BAS' = 6 ; MIL' = (INT (HAUT + BAS 1/2) = 7 MIL = PAPER on a trouvé le NOM cherché en 2 coups au lieu de 7 dans le cas d'un balayage séquentiel. Maintenant la lecture de INDEX (7) donne le numéro de l'enregistrement INDEX (7) = 9.

Le programme basic suivant met en œuvre cette méthode en outre les enregistrements blancs (non utilisés) sont supprimés de la table.

La table est classée à l'aide du tri à bulle, mais n'importe qu'elle autre méthode conviendrait aussi.

INDEXS

```
10 '
20 '**********************
30 '* ACCES INDEXE SIMPLE *
40 '**********************
50 '
60 J=1:NO$="":PN$="":AD$=""
70 ?OPEN "QWERT=1"
80 ?WHERE1
90 NE=PEEK(#48F)
100 DIM NO$(NE),INDEX(NE)
120 FOR I=1 TO NE
130 ?TAKE1,I'NO$
140 IF NO$=CHR$(2) THEN GOTO 180
150 NO$(J)=NO$
160 INDEX(J)=I
170 J=J+1
180 NEXT I
185 J=J-1
190 '
490 '*************************
500 '* CLASSEMENT DE LA TABLE *
510 '*       TRI A BULLE       *
520 '*************************
530 '
540 FOR I=1 TO J-1
550 FOR K=I+1 TO J
560 IF NO$(I)<=NO$(K) THEN GOTO 580
570 X$=NO$(I):NO$(I)=NO$(K):NO$(K)=X$
575 X=INDEX(I):INDEX(I)=INDEX(K):INDEX(
K)=X
580 NEXT K
590 NEXT I
```

```
810 '*********************
820 '* RECHERCHE D'UN NOM *
830 '*********************
840 '
850 INPUT"NOM: ";CLE$
860 M$="NOM INEXISTANT"
870 HAUT=J:BAS=1
880 IF BAS>HAUT THEN PRINT M$:GOTO 850
890 MIL=INT((BAS+HAUT)/2)
900 IF NO$(MIL)=CLE$ THEN GOTO 1000
910 IF CLE$>NO$(MIL)THENBAS=MIL+1 ELSE
HAUT=MIL-1
920 GOTO 880
950 '
960 '
1000 EC=INDEX(MIL)
1010 ?TAKE1,EC'NO$,PN$,NR%,AD$
```

Dans ce programme, la gestion de suppression d'un enregistrement n'est pas prise en compte. Autrement dit, on supprime un élément en procédant de la manière classique, mais l'élément n'est pas supprimé de la table d'index. Il le sera à la prochaine ouverture du fichier. L'inconvénient de ce type de méthode est qu'on ne peut pas récupérer un enregistrement effacé au cours de la même session car rien n'indique dans l'index qu'il est libre.

2° - ACCES INDEXE AVEC SAUVEGARDE DE LA TABLE D'INDEX

Dès que le fichier devient important, (taille supérieure à 200 éléments) nous avons vu que la table d'index peut mettre plusieurs minutes à se constituer (surtout avec les méthodes de tri les plus simples). Pour les fichiers importants qui sont souvent modifiés (fichiers de mouvements) la solution précédante n'est plus acceptable. D'autres méthodes ont donc été mises au point. Le principe est le suivant : plutôt que de créer la table d'index à chaque session, elle est mémorisée, il n'y a donc qu'à la recharger. Cette méthode séduisante nécessite de prendre des précautions par-

ticulières. En effet, il s'agit maintenant de gestion en « temps réel » puisque la table qui a été créée à la naissance du fichier doit durer jusqu'à sa mort. Il n'est donc plus question de différer la prise en compte de l'effacement d'un enregistrement. Par ailleurs, on n'est pas à l'abri d'une coupure d'électricité qui interviendrait en cours de traitement ; il faut pouvoir reprendre le travail en cours : donc il faut se ménager la possibilité de reconstituer la table d'index en cas de problème.

Cette méthode présente un avantage : la gestion des enregistrements libres est dynamique. Quand un enregistrement n'est plus utilisé (quand il est effacé), il est susceptible d'être aussitôt réutilisé.

Le programme suivant montre la méthode à appliquer. La création du fichier n'est pas traitée, on procède comme nous avons vu jusqu'à présent (il ne faut pas oublier l'initialisation avec des codes ASCII 2).

Ensuite, il faut appeler la procédure de création de la table d'index : le fichier est lu séquentiellement, le premier champ de chaque enregistrement est placé dans la table d'index N0$, le numéro d'enregistrement dans la table INDEX.

L'index N0$ est ensuite trié de telle sorte que les noms soient par ordre alphabétique au sommet de la table, suivis par les enregistrements libres. Ceci étant, les deux tables d'index sont sauvées (! MSAVE...).

Notons tout de suite que c'est aussi cette routine qui permet de remettre la table d'index en ordre au cas où des incohérences seraient constatées (fausses manœuvres, coupure d'électricité...). Il s'agit d'une routine de secours donc le programme principal ne permet pas d'y accéder il faut taper RUN 1150.

Programme principal : il faut tout d'abord charger les tables d'index en mémoire centrale (lignes 50-120). Les lignes 120-180 sont facultatives, elles permettent d'afficher le contenu des index à titre expérimental.

Recherche d'un nom : comme précédemment la recherche d'un élément s'effectue par dichotomie (S/Prog. 1020) les lignes 310-390 sont un exemple de la gestion des doubles, la recherche dichotomique permet de tomber sur un des éléments cherchés, pas forcément le premier ou le dernier, donc quand on a trouvé l'un des éléments cherchés on se déplace vers le haut et vers le bas pour chercher les autres (qui sont forcément contigus, la table étant classée).

INSERTION D'UN ELEMENT : on recherche le premier emplacement libre en lisant séquentiellement N0$ (), si on n'en trouve pas le fichier est plein, sinon la lecture INDEX nous indique quel est son numéro d'enregistrement. Il est aussitôt occupé par la nouvelle fiche. Il faut donc remettre maintenant l'index à jour. On profite de ce qu'il est classé pour insérer la fiche directement à sa place de la manière suivante.

	NO$	INDEX
	ARN	7
	BIQ	5
	REN	8
	WOF	1
I➧	2	9
	2	10
EC = 9	2	:

CLE = MON on veut insérer MON

1 on compare WOF et MON. WOF est supérieur à MON donc on descend WOF d'un cran dans N0$
2 on compare MON et REN. REN est supérieur à MON donc on descend REN d'un cran
3 on compare BIQ et MON. BIQ est inférieur à MON on insère MON et EC.

Après 1°

	NO$	INDEX
	ARN	7
	BIQ	5
	REN	8
I➧	WOF	1
	WOF	1
	2	10

	NO$	INDEX
	ARN	7
	BIQ	5
	REN	8
I➧	REN	8
	WOF	1
	2	10

Après 2°

	NO$	INDEX
	ARN	7
	BIQ	5
I➧	MON	9
	REN	8
	WOF	1
	2	10

Après 3°

Après un ajout l'index est immédiatement classé.

SUPPRESSION D'UN ELEMENT :

Pour supprimer un élément on recherche avec la clé comme pour une lecture ; ensuite on écrit ASCII 2 dans le premier champ de l'enregistrement Index (I).

SAVINDEX

```
10 '*********************************
20 '* ACCES INDEXE AVEC SAUVEGARDE *
30 '*********************************
40 '
50 J=1:NO$="":PN$="":AD$=""
60 ?OPEN "QWERT=1"
70 ?WHERE1
80 NE=PEEK(#48F)
90 DIM NO$(NE),INDEX(NE)
100 ?MLOAD"NOM$=NO$"
110 ?MLOAD"INDEX=INDEX"
120 '
130 FORI=1TO NE
140 PRINT NO$(I),INDEX(I)
150 NEXT
160 STOP
170 CLS
180 PRINT"RECHERCHER(R),AJOUTER(A),SUPP
RIMER(S)     FERMER(F)"
190 PRINT@20,4;"";
200 GET A$
210 IF A$="A" THEN GOTO 460
220 IF A$="R" THEN GOTO 250
230 IF A$="S" THEN GOTO 740
240 IF A$="F" THEN GOTO  930 ELSE GOTO
200
250 '
260 '**********************
270 '* RECHERCHE D'UN NOM *
280 '**********************
290 '
300 GOSUB1020
310 IF NO$(MIL-1)=CLE$ THEN MIL=MIL-1 G
```

```
OTO 310
 320 REPEAT
 330 EC=INDEX(MIL)
 340 !TAKE1,EC'NO$,PN$,AD$
 350 MIL=MIL+1
 360 PRINTNO$,PN$
 370 PRINTAD$
 380 WAIT 300
 390 UNTIL NO$(MIL)<>CLE$
 400 GOTO 170
 410 '
 420 '****************************
 430 '* AJOUT D'UN ENREGISTREMENT *
 440 '****************************
 450 '
 460 INPUT"NOM: ";NO$
 470 INPUT"PRENOM: ";PN$
 480 INPUT"ADRESSE: ";AD$
 490 I=1:FL=0
 500 REPEAT
 510 IF NO$(I)=CHR$(2) THEN FL=1
 520 I=I+1
 530 UNTIL I=NE+1 OR FL=1
 540 IF I=NE+1 AND NO$(I-1)<>CHR$(2) THE
N PRINT"FICHIER PLEIN":WAIT300:GOTO170
 550 I=I-1
 560 EC=INDEX(I)
 570 !WRITE1,EC'NO$,PN$,AD$
 580 '
 590 FL=0
 600 REPEAT
 610 IF NO$>NO$(I-1) THEN FL=1:GOTO 650
 620 NO$(I)=NO$(I-1)
 630 INDEX(I)=INDEX(I-1)
 640 I=I-1
```

```
650 UNTIL FL=1 OR I=1
660 NO$(I)=NO$
670 INDEX(I)=EC
680 GOTO170
690 '
700 '***************
710 '* SUPPRESSION *
720 '***************
730 '
740 GOSUB 1020
750 A$=CHR$(2)
760 ?WRITE 1,EC'A$
770 I=MIL
780 X=INDEX(MIL)
790 IF I=NE THEN GOTO 850
800 REPEAT
810 NO$(I)=NO$(I+1)
820 INDEX(I)=INDEX(I+1)
830 I=I+1
840 UNTIL NO$(I)=CHR$(2) OR I=NE
850 IF I=NE THEN NO$(NE)=A$:INDEX(NE)=E
C:GOTO 870
860 NO$(I-1)=A$:INDEX(I-1)=EC
870 GOTO 170
880 '
890 '************************
900 '* FERMETURE DU FICHIER *
910 '************************
920 '
930 ?MSAVE"NOM$=NO$"
940 ?MSAVE"INDEX=INDEX"
950 ?CLOSE0
960 END
970 '
```

```
980 '*************************
990 '* LECTURE DANS LE FICHIER *
1000 '*************************
1010 '
1020 I=0
1030 REPEAT
1040 I=I+1
1050 UNTIL NO$(I)=CHR$(2) OR I=NE
1060 INPUT"NOM: ";CLE$
1070 M$="NOM INEXISTANT"
1080 HAUT=I:BAS=1
1090 IF BAS>HAUT THEN PRINT M$:GOTO 106
0
1100 MIL=INT((BAS+HAUT)/2)
1110 IF NO$(MIL)=CLE$ THEN EC=INDEX(MIL
):RETURN
1120 IF CLE$>NO$(MIL) THEN BAS=MIL+1 EL
SE HAUT=MIL-1
1130 GOTO 1090
1140 '
1150 '****************************
1160 '* MISE A JOUR TABLE D'INDEX *
1170 '****************************
1180 '
1190 NO$=""
1200 !OPEN"QWERT=1"
1210 !WHERE1
1220 NE=PEEK(#48F)
1230 STOP
1240 DIM INDEX(NE),NOM$(NE)
1250 FOR I=1 TO NE
1260 !TAKE1,I'NO$
1270 NO$(I)=NO$
1280 INDEX(I)=I
1290 NEXT I
1300 NO$(0)=CHR$(2)
```

```
1310 INDEX(0)=0
1320 '
1330 '*************************
1340 '* CLASSEMENT DE LA TABLE *
1350 '*       TRI A BULLE      *
1360 '*************************
1370 '
1380 FOR I=1 TO NE-1
1390 FOR K=I+1 TO NE
1400 IF NO$(I)<=NO$(K) THEN GOTO  1430
1410 X$=NO$(I):NO$(I)=NO$(K):NO$(K)=X$
1420 X=INDEX(I):INDEX(I)=INDEX(K):INDEX
(K)=X
1430 NEXT K
1440 NEXT I
1450 I=0
1460 REPEAT
1470 I=I+1
1480 UNTIL NO$(I)<>CHR$(2)
1490 I=I-1
1500 FOR J=1 TO NE-I
1510 X$=NO$(J):NO$(J)=NO$(J+I):NO$(J+I)
=X$
1520 X=INDEX(J):INDEX(J)=INDEX(J+I):IND
EX(J+I)=X
1530 NEXT J
1540 !MSAVE"NOM$=NO$"
1550 !MSAVE"INDEX=INDEX"
1560 STOP
1570 '
```

Il ne faut pas oublier de remettre à jour les tables d'index. La méthode est celle inverse de l'insertion : il faut boucher le trou :

On veut supprimer MON ⧫ EC = 9 suppression sur le disque.

ARN 7	ARN 7	ARN 7	ARN 7
BIQ 5	BIQ 5	BIQ 5	BIQ 5
I ⧫ MON 9	I ⧫ REN 8	REN 8	REN 8
REN 8	REN 8	I ⧫ WOF 1	WOF 1
WOF 1	WOF 1	WOF 1	I ⧫ 2 9
2 10	2 10	2 10	2 10

On voit que l'enregistrement 9 est libéré, il pourra être réutilisé au prochain ajout.

FERMETURE DU FICHIER : il ne faut pas oublier de sauver les tables d'index, car celles sur disque ne sont peut-être plus à jour.

3° - CALCUL DU N° D'ENREGISTREMENT A PARTIR DE LA CLE : HASH-CODING

Quand les tables d'index deviennent trop longues, elles occupent une place en mémoire centrale considérable. Si on juge cette perte trop pénalisante, on a recours au Hash-Code. Le principe de cette méthode est de calculer le numéro d'enregistrement à partir de la clé. On définit donc une fonction mathématique qui tient compte des propriétés de la clé et qui fournit un nombre compris entre 1 et le nombre total d'enregistrements prévus. Cette fonction doit fournir des nombres aléatoires, qui sont répartis au mieux sur l'intervalle alloué.

Si la clé est numérique, une bonne méthode consiste à :
- l'élever au carré
- prendre les 4 chiffres du milieu + 1
- prendre l'inverse on obtient un nombre compris entre 0 et 1
- le n° d'enregistrement est alors donné par ce nombre x par le nombre total d'enregistrements du fichier.

Si la clé est alphanumérique on fait correspondre à chaque symbole un nombre (son code ASCII par exemple, sa position dans l'alphabet...).

Dans le programme exemple nous avons utilisé la fonction suivante : on a tout d'abord supposé que tous les noms avait plus de trois lettres.

Dans la clé, on a extrait les 4 lettres du milieu, à chaque lettre on a fait correspondre son code ASCII.

La clé numérique est donnée par la suite d'opération
CL = CODE 1re lettre x 997 + CODE 2e lettre x 97 + CODE 3e lettre x 7 + CODE 4e lettre x 13.
On prend le log décimal de CL et on garde comme clé le 3e et 4e chiffre décimal augmenté de 1.

Pour insérer un enregistrement, il suffit de calculer la clé et de placer les informations.

Pour récupérer un enregistrement, on calcule de nouveau la clé et on accède ainsi directement aux informations.

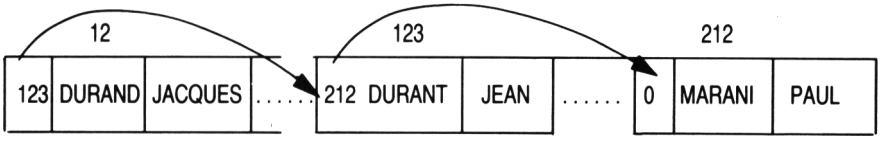
Cette technique exploite directement les possibilités d'un fichier à accès direct.

Malheureusement, il peut arriver que deux clés d'origine correspondent à deux clés calculées identiques. Ce phénomène appelé « collision », est d'autant plus fréquent que le taux de remplissage du fichier est important. Il faut donc tenir compte de ces « accidents ».

Tout d'abord on réduit le risque en consentant à perdre une place sur le fichier de 20 à 30 %.

Ensuite, il faut tout de même étudier un processus de gestion des collisions. La méthode la plus simple est celle employée dans l'exemple. Chaque enregistrement est précédé d'un pointeur, si ce pointeur est = 0, il n'y a pas encore eu de collision, s'il n'est pas à zéro, il indique l'adresse de l'enregistrement jumeaux

par exemple



On a ainsi créé une structure de liste (simple chaînage, en fait on a simulé le principe du fichier à accès séquentiel).

HASH

```
10 '**************
20 '* HASCH-CODE *
30 '**************
40 '
50 NO$="":PN$="":AD$="":A$=""
60 ?OPEN"HASH=1"
70 CLS
80 PRINT"RECHERCHER(R),AJOUTER(A),SUPPR
IMER(S)        FERMER(F)"
90 PRINT@20,4;"";
100 GET A$
110 IF A$="A" THEN GOTO 390
120 IF A$="R" THEN GOTO 710
130 IF A$="S" THEN GOTO  880
140 IF A$="F" THEN?CLOSE1:END
150 GOTO100
160 '
170 '********************
180 '* CALCUL DE LA CLE *
190 '********************
200 '
210 HA=INT(LEN(NO$)-4)/2+1:IF HA<1 THEN
HA=1
220 HA$=MID$(NO$,HA,4)
230 FORI=1 TO 4
240 HA(I)= ASC(MID$(HA$,I,1))
250 NEXTI
260 CL=997*HA(1)+997*HA(2)+97*HA(3)+13*
HA(4)
270 CL=LOG(CL) :PRINTCL
280 CL=CL-INT(CL):PRINTCL
290 CL=CL*100:PRINTCL
300 CL=CL-INT(CL):PRINTCL
```

```
310 CL=CL*100:PRINTCL
320 CL=INT(CL)
330 RETURN
340 '
350 '*****************************
360 '* AJOUT D'UN ENREGISTREMENT *
370 '*****************************
380 '
390 INPUT"NOM: ";NO$
400 INPUT"PRENOM: ";PN$
410 INPUT"ADRESSE: ";AD$
420 GOSUB 210
430 ?TAKE1,CL'PT,A$
440 IF A$<>CHR$(2) THEN GOTO 480
450 PT=0
460 ?WRITE1,CL'PT,NO$,PN$,AD$
470 GOTO 70
480 IF PT=0 THEN M=CL:GOTO 530
490 REPEAT
500 M=PT
510 ?TAKE1,PT'PT
520 UNTIL PT=0
530 I=100
540 REPEAT
550 ?TAKE1,I'PT,A$
560 I=I+1
570 UNTIL I=151 OR A$=CHR$(2)
580 IF I=151 THEN PRINT "FICHIER PLEIN"
:WAIT300:GOTO 70
590 I=I-1
600 PT=0
610 ?WRITE1,I'PT,NO$,PN$,AD$
620 ?TAKE1,M'PT,NO$,PN$,AD$
630 PT=I
640 ?WRITE1,M'PT,NO$,PN$,AD$
650 GOTO 70
```

```
660 '
670 '****************************
680 '* RECHERCHE ENREGISTREMENT *
690 '****************************
700 '
710 INPUT"NOM: ";NO$
720 GOSUB 210
730 !TAKE1,CL'PT,A$,PN$,AD$
740 IF NO$=A$ THEN GOTO 780
750 IF PT=0 THEN GOTO 810
760 CL=PT
770 GOTO 730
780 PRINTA$,PN$
790 PRINTAD$:WAIT300
800 GOTO 70
810 PRINT"NOM INCONNU ":WAIT 300:GOTO 7
0
820 GOTO 70
830 '
840 '************************
850 '* EFFACE ENREGISTREMENT *
860 '************************
870 '
880 INPUT"NOM: ";NO$
890 GOSUB 210
900 !TAKE1,CL'PT,A$
910 IF A$=NO$ AND PT=0 THEN GOTO  960
920 IF A$=NO$ AND PT<>0 THEN GOTO 1000
930 IF A$<>NO$ AND PT<>0 THEN GOTO 1060
940 PRINT"NOM INCONNU":WAIT 300
950 GOTO 70
960 A$=CHR$(2)
970 !WRITE1,CL'PT,A$
980 GOTO 70
990 CL=PT
```

```
1000 ?TAKE1,PT'P1,NO$,PN$,AD$
1010 ?WRITE1,CL'P1,NO$,PN$,AD$
1020 P1=0
1030 A$=CHR$(2)
1040 ?WRITE1,PT'P1,A$
1050 GOTO 70
1060 CL=PT
1070 GOTO  900
1080 '
1090 '************************
1100 '*INITIALISATION FICHIER *
1110 '************************
1120 '
1130 ?OPEN"HASH=1"
1140 ?WHERE1
1150 NE=PEEK(#48F)
1160 A$=CHR$(2):PT=0
1170 FOR I=1 TO NE
1180 ?WRITE1,I'PT,A$
1190 NEXT I
1200 STOP
```

Programme : commentaires (voir listing).

Pour utiliser ce programme, il faut d'abord créer un fichier de nom HASH, d'une dimension de 120 à 150 enregistrements. Les 100 premiers constituent le fichier principal, les autres forment le fichier des collisions. On fait RUN 130 pour initialiser ce fichier, le premier champ est numérique et rempli de zéros, tandis que le second est alphanumérique et rempli de codes ASCII 2.

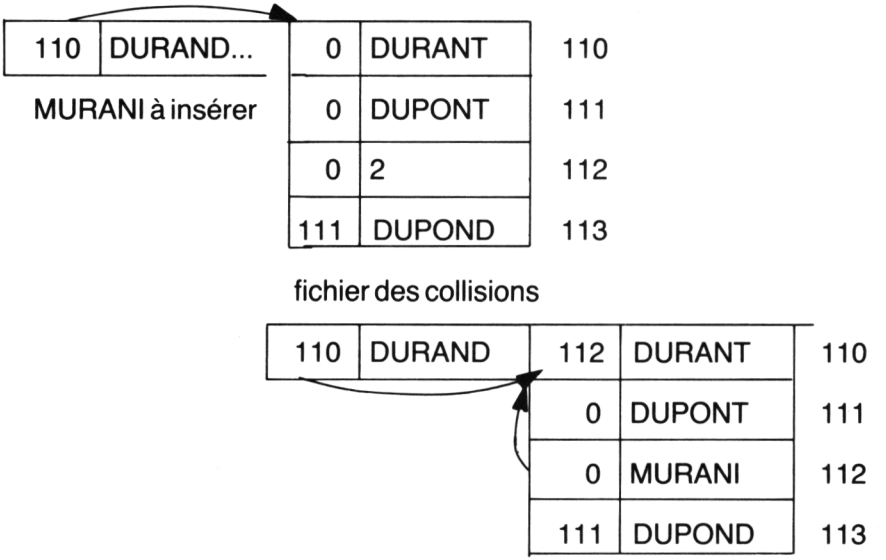
Revenons maintenant au début du programme.

Lignes 50 à 150 : Initialisation et menu.

Lignes 210 à 330 : Sous programme de calcul de la clé : elle est calculée comme expliqué précédemment, des PRINT permettent de suivre le déroulement de ce calcul.

380 - 650 : Ajout d'un enregistrement.

Pour ranger un enregistrement, on calcule tout d'abord sa clé numérique. Ensuite, on vérifie que l'enregistrement est libre : c'est le cas si A$ = CHR$ (2). On peut y écrire ; si A$ ‹› CHR$ (2) on s'intéresse à la variable PT. Si elle est égale = 0 il n'y a pas encore eu de collision sur ce numéro de clé, sinon on lit l'enregistrement dont le numéro est donné précisément par PT, jusqu'à rencontrer le premier enregistrement ou PT = 0 on mémorise son numéro, ensuite on parcours séquentiellement le fichier des collisions jusqu'à trouver une place libre, on y range l'enregistrement et on remet à jour le pointeur de chaînage de l'enregistrement mémorisé :



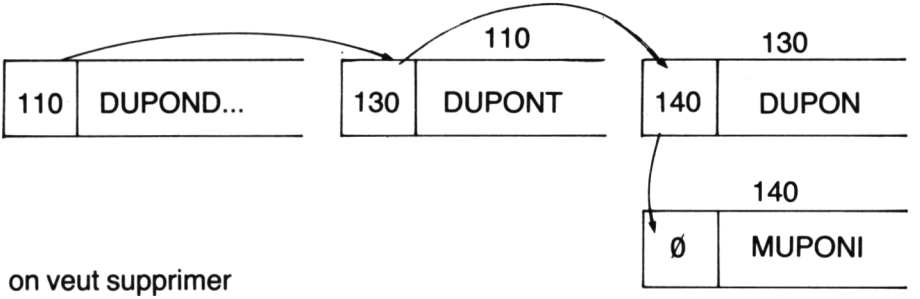
La lecture du fichier montre que l'enregistrement n° 112 est libre. Donc on y range « MURANI » et on établit le chaînage.

Lignes 710-820 : Recherche d'un enregistrement.

On calcule la clé, puis on lit l'enregistrement. Si la clé alphanumérique et le nom lu concordent on a trouvé le bon enregistrement et on l'affiche, sinon la variable PT fournit le nom du prochain enregistrement à lire. Si PT vaut 0 et que les clés ne sont pas accordées alors l'enregistrement n'existe pas.

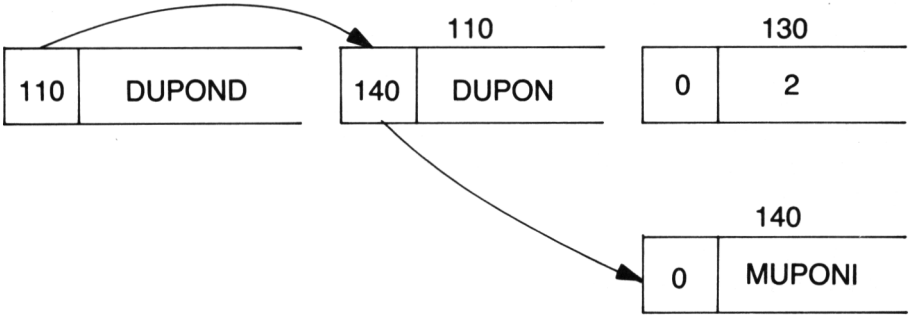
Lignes 880-1070 : Effacement d'un enregistrement.

Le principe est très simple : si la variable PT est à zéro, on remplit les deux premiers champs avec respectivement 0 et CHR$ (2), si la variable PT n'est pas à zéro on procède comme suit :



on veut supprimer

DUPONT on l'écrase avec l'enregistrement qui lui est chaîné, et on efface l'enregistrement ainsi copié.



L'enregistrement ainsi libéré peut être utilisé pour d'autres collisions.

LANGAGE MACHINE

Introduction à l'utilisation du lecteur de disquette à partir du langage machine.

Ce chapitre est destiné aux inconditionnels du langage machine qui désirent utiliser le lecteur de disquettes dans leurs propres programmes. Il ne comporte donc pas de rappel sur la manière d'utiliser le langage machine du 6502 et sur l'ORIC (se reporter à d'autres livres, il en existe de nombreux).

I - LA PREMIERE OPERATION EST D'APPRENDRE A MANIPULER LA RAM OVERLAY :

Les 16 kOctets de RAM en overlay sont situés en parallèle avec la ROM Basic de l'ORIC. On ne peut donc pas utiliser simultanément l'un et l'autre ; des broches sur le connecteur d'extension (MAP et ROMDIS) permettent d'effectuer la sélection ROM-RAM.

Le DOS est logé dans la RAM Overlay pour ne pas occuper de place sur la mémoire « normale » utilisateur. La carte contrôleur du lecteur permet donc d'accéder à la RAM Overlay. Le procédé est même très simple pour l'utilisateur.

- Pour accéder à la RAM il faut

interdire les interruptions : SEI ; LDA % # 7 F ; STA # 30E
mettre 1 dans la position # 3FA : LDA # 01
STA # 03FA

- Pour remettre les choses dans leur état normal (accéder à la ROM) il faut mettre « 0 » dans # 3FA : LDA% # 00
STA # 3FA
puis autoriser les interruptions : LDA% # C0,STA # 30 E,CLI.

A titre d'exemple, nous donnons ci-dessous un utilitaire permettant de transférer dans la RAM utilisateur des parties du DOS pour pouvoir les analyser ensuite :

```
TRANSFER

 10 AD=#B800
 20 'ADRESSE DE DEPART-->#50,#51
 30 'ADRESSE DE FIN   -->#52,#53
 40 'ADRESSE D'ARRIVEE-->#54,#55
 50 REPEAT
 60 READ A
 70 POKEAD,A
 80 AD=AD+1
 85 UNTIL AD=#B800+46
 90 INPUT"ADRESSE DE DEPART ZONE A TRANS
FERER";D
 95 DOKE#50,D
 100 INPUT"ADRESSE FIN ZONE A TRANSFERER
";F
 105 DOKE #52,F
 110 INPUT"ADRESSE DE RELOGEMENT";R
 115 DOKE #54,R
 120 IF D>F THEN PRINT"ERREUR":GOTO 90
 125 CALL #B800
 126 END
 200 DATA #78:'SEI
 205 DATA #A9:'LDA %#01
 206 DATA #01
 207 DATA #8D:'STA #03FA
 208 DATA #FA
 209 DATA #03
 210 DATA #A0:'LDY %#00
```

```
220 DATA #00
230 DATA #A5:'LDA #50
240 DATA #50
250 DATA #C5:'CMP #52
260 DATA #52
270 DATA #D0:'BNE #06
280 DATA #06
290 DATA #A5:'LDA #51
300 DATA #51
310 DATA #C5:'CMP #53
320 DATA #53
330 DATA #F0:'BEQ #13
340 DATA #13
350 DATA #B1:'LDA (#50),Y
360 DATA #50
370 DATA #91:'STA (#54),Y
380 DATA #54
390 DATA #E6:'INC #50
400 DATA #50
410 DATA #D0:'BNE #02
420 DATA #02
430 DATA #E6:'INC #51
440 DATA #51
450 DATA #E6:'INC #54
460 DATA #54
470 DATA #D0:'BNE #02
480 DATA #02
490 DATA #E6:'INC #55
500 DATA #55
510 DATA #4C:'JMP #B800
520 DATA #00
530 DATA #B8
540 DATA #A9:'LDA %#00
550 DATA #00
560 DATA #8D:'STA #03FA
```

```
570 DATA #FA
580 DATA #03
590 DATA #58:'CLI
600 DATA #60:'RTS
```

Commentaires :

- Le progamme en l.m. est fourni par les DATA ; les lignes 40 à 90 permettent l'implantation du programme dans le 2e clavier de l'ORIC. Pour le reloger il suffit de modifier les lignes JMP#B808 (550,560,570).

- L'adresse de départ de la zone à transférer : c'est la première adresse du bloc que l'on veut transférer.

- L'adresse de fin de la zone à transférer : c'est la dernière adresse du bloc à transférer.

- L'adresse de relogement : c'est l'adresse où le premier octet du bloc translaté sera placé.

- Fonctionnement du programme en langage machine.

Lignes 190-240 : autorisation d'entrée dans l'overlay.

Lignes 250-260 : initialisation de Y.

Lignes 270-380 : On compare les adresses de début et de fin de bloc : si elles sont égales retour au basic sinon...

Lignes 390-420 : déplacer un octet.

Lignes 430-570 : incrémenter adresse départ et adresse du relogement.

Puis recommencer au début.

Lignes 580-640 : Remise en état pour accéder de nouveau en basic, puis retour au basic.

Carte mémoire du TDOS :
Adresses
C000-C0FF : adresses utilisées par le DOS
C100-E0FF : programme du TDOS
E100-E120 : liste des mots types (BAS,BIN...)
E120-E25F : liste des mots clés (CAT, WRITE)
E260-E47F : liste des messages d'erreurs, et des messages divers
E480-E4FF : table des jumps dans l'ordre des mots clés (voir tableau)

E500-E5FF : programme du BOOT
E600-E6FF : programme assurant le lien avec la page 4
E700-E7FF : sauvegarde de la pile de l'Oric (page 1 de la RAM)
E800-EBFF :
EC00-ECFF : buffer liste piste secteur en entrée
ED00-EDFF : buffer liste piste secteur en sortie
EE00-EEFF : Bit Map disc 1 (Buffer)
EF00-EFFF : Bit Map disc 2 (Buffer)
F000-F1FF : libre (utilisé par le DOS)
F200-F2FF : Buffer directory (catalogue)
F300-FBFF : utilisé par DOS
FC00-FDFF : Buffer « Fichiers ouverts
FE00-FEFF : Buffer Entrée/Sortie

Table des correspondance :

WRITE	E3F2
TAKE	E3F8
JUMP	F4DB
APND	E0F8
CUT	F167
REWIND	DCBA
OPEN	DD84
WL	DE24
WUL	DE1C
CREATE	DA4A
CLOSE	DCA7
MSAVE	F835
MLOAD	F87E
SAVE	C9BA
LOAD	C445
CAT	D48B
LCAT	D49A
RENAME	D243
COPY	D635
DEL	C4B9
LOCK	C4E6
UNLOCK	C4DE
START	F5BE
UNSTART	F5D9
MOUNT	D621
DENAME	D68D

INIT	C2CD
DEMOUNT	D3CD
ERSET	DEB2
EROFF	DEBA
WHERE	F4BF
SEARCH	CFBA
MERGE	F0A5
HSCR	E404
LSCR	E40A
MASTER	CEC1

La liste des mots types : elle comprend les mots suivants : BAS,BIN,SYS,CMD,TXT,DAT,ARY,SCR

Liste des mots clés et table des jumps :

Ces deux tables permettent de connaître l'adresse de début de chacune des instructions du DOS. Ceci peut vous être utile pour désassembler des parties précises ou pour utiliser le lecteur par un programme en langage machine.

L'adresse du n^ième^ saut correspond à la nième instruction de la liste des mots. Pour éviter des recherches fastidieuses nous donnons les correspondances dans le tableau ci-après. Pour utiliser les fonctions du lecteur en langage machine, il suffit de placer à partir de l'adresse # 35 le nom de la fonction suivi de ses attributs, puis d'appeler le sous-programme situé en # 400. (voir programme ci-dessous).

```
>Z1000,1017
1000 A0FF    LDY %#FF
1002 C8      INY
1003 B93010  LDA #1030,Y
1006 993500  STA #0035,Y
1009 D0F7    BNE #1002
100B A935    LDA %#35
100D 85E9    STA #E9
100F A900    LDA %#00
1011 85EA    STA #EA
1013 200004  JSR #0400
1016 60      RTS
>L
```

```
>U1030,1049

1030 4C 4F 41 44 22 43 4C 41 LOAD"CLA
1038 53 43 41 54 2E 42 49 4E SCAT.BIN
1040 22 00 55 55 55 55 55 55 ".UUUUUU
1048 55
>L
```

Remarque :

L'octet 1042 = 00 est nécessaire : il sert de « butée » à la boucle du programme, il est testé à la ligne 1009.

Buffers de BITMAP

Le contenu de ces buffers indique quels sont les secteurs libres et les secteurs occupés sur la disquette.

Chaque piste est représentée par 3 octets (24 bits) un bit à zéro indique que le secteur est occupé, un bit à 1 indique un secteur libre.

La représentation des secteurs d'une piste est la suivante :

1° octet	2° octet	3° octet
10 11 12 13 14 15 16 17	2 3 4 5 6 7 8 9	L x x x x x x 1

n° secteur

x : innoccupé

L : si L = 1 toute la piste est occupée
si L = 0 il reste des secteurs libres sur la piste

Les octets inoccupés sont placés à la valeur # 80. Les 8 derniers octets du Buffer représentent le nom du volume. La Bit Map est bien évidemment mémorisée sur le disque elle se trouve sur la piste 20 (décimal ; 14 H) secteur 1.

Buffer Catalogue (Buffer directory)

Le catalogue de la disquette est chargé dans ce buffer. Le contenu de ce buffer permet au DOS de retrouver un programme contenu sur le disque.

Le contenu du Buffer est le suivant :

Octet
0,1 : FF si disque non présent ou lecteur non monté sinon
- 00 00 pour le premier secteur du catalogue ;
- pour les secteurs suivants ces octets représentent respectivement le numéro de piste et le numéro de secteur où le bloc est stocké sur le disque.

Le premier bloc du catalogue est toujours placé piste # 14, secteur 2 ; les autres secteurs sont alloués en fonction de la place disponible sur le disque ce qui explique qu'il soit nécessaire de mémoriser leur place (pour les modifier ou les remettre à jour).

Octet 2,3 : numéro de piste et de secteur du prochain bloc catalogue dans le cas où le catalogue complet est plus long qu'un secteur. (C'est ce qu'on appelle le lien de chaînage) le reste du secteur est constitué de 14 segments de 8 octets permettant de décrire 14 fichiers. (Si le catalogue comporte plus de 14 titres il occupe plus d'un secteur, ce qui se vérifie facilement en demandant ! CAT «». Le disque se met à tourner, la première liste est affichée puis le lecteur recommence à tourner et les 14 titres suivants sont affichés.... etc.)

Les 18 octets descriptifs de chaque fichier se répartissent comme suit :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
N	S	U L	N	O	M	△	△	△	△	△	•	B B	I A	N S	S D	L	L

Les 2 premiers octets : numéro de piste et de secteur de la liste pistes/secteurs du fichier (File control Block ou FCB), ou FF si le fichier est supprimé du disque).
Octet 2 : code ASCII de U si le fichier n'est pas verrouillé (# 55)
code ASCII de L si le fichier est verrouillé.

Octets 3 à 10 : nom du fichier limité à 8 lettres, si le nom comporte moins de 8 lettres les octets vides sont remplis avec des espaces (code ASCII # 20)
Octet 11 : code ASCII du «.»
Octets 12,13,14 : Code ASCII des 3 lettres de l'extension BIN,BAS,TXT.
Octet 15 : type du fichier séquentiel ou direct S ou D
Octets 16-17 : nbres de secteurs occupés par le fichier sur le disque codé en hexadécimal (dans le cas d'un fichier de plus de 256 secteurs le poids faible est sur l'octet 16, le poids fort sur l'octet 17).

Buffers listes pistes/secteurs (en entrée et en sortie).

Ces buffers contiennent les listes pistes/secteurs d'un fichier. Dans le cas d'une opération d'écriture (WRITE, SAVE...) c'est le buffer entrée qui est chargé. Dans le cas d'une opération de lecture c'est le buffer sortie qui est rempli.

Il existe trois types de listes pistes/secteurs :
1 - les FCB fichiers programme
2 - les FCB fichiers à accès séquentiel
3 - les FCB fichiers à accès direct

1 - FICHIERS PROGRAMME

Les secteurs sont découpés comme suit.

Octets 0,1 : lien de chaînage : numéro de piste et de secteur de la suite de la liste pistes/secteurs. Si l'octet 0 = FF et le suivant = # 00 pas de chaînage.

Octets n° 2,3 : Contiennent l'adresse de début de chargement du programme codée en hexadécimal, l'octet de poids faible en 2, l'octet de poids fort en 3.

Octets 4,5 : longueur totale du fichier (en octets) codée en hexa suivant la même disposition que précédemment. Les 250 octets restants sont regroupés par 2 et représentent ces numéros de piste et de secteur où le programme est stocké sur le disque. (Octets 6,7 : n° piste/secteur du 1er secteur, octects 8,9 : n° piste/secteur du 2e secteur, octets 254,255 n° piste (secteur du 125° secteur de programme). Quand ils ne sont pas utilisés, ces octets contiennent la valeur FF.

2° - FICHIERS A ACCES SEQUENTIEL

Le FCB est contenu dans l'enregistrement n° 0 du fichier (ce qui explique qu'il n'est pas disponible pour l'utilisateur). Une fiche est représentée sur le disque par une suite de codes. ASCII image. de chacun des caractères de la fiche.

Chaque enregistrement est séparé du suivant par un code ASCII # FF, le FCB est séparé du premier enregistrement par le code # FF, le dernier enregistrement est marqué par les deux caractères : # 00 suivit de # F0.

Le FCB est logé pendant l'ouverture du fichier dans le buffer fichier ouvert.

La structure du FCB est la suivante :
Octet 0 : indique le numéro logique du fichier
Octet 1 : numéro du lecteur.
Octet 2 : Type du fichier ; pour un fichier à accès séquentiel, il est toujours égal à 1.
Octet 3 : Protection du fichier ; 0 = WUL 1 = WL
Octets 4,5 : numéro de piste et de secteur du FCB (si fichier vide) ou du premier enregistrement (dans le cas contraire).
Octets 6,7 : inutilisés pour les fichiers à accès séquentiel.
Octets 8,9 : nombre total d'enregistrements : numéro du dernier enregistrement.

Octets 10,11 : numéro de l'enregistrement courant. Attention : Après une lecture l'enregistrement courant est le successeur de l'enregistrement lu.
Octets 12,13 : numéro de piste et de secteur de la première liste pistes/secteurs du fichier.
Octets 14,15,16 : numéro de piste et de secteur de la liste pistes/secteurs exploitée et déplacement dans cette liste.
Octets 17,18,19 : idem que 14,15,16 mais pour la dernière liste pistes/secteurs.
Octets 20,21 : numéro de piste et de secteur du premier enregistrement.
Octets 22,23,24 : numéro de piste, de secteur et déplacement pour l'enregistrement courant.
Octets 25,26,27 : idem Octets 22,23,24 mais pour le dernier enregistrement.

3° - FCB FICHIERS A ACCES DIRECT

Les fiches sont séparées par un octet à FF, les rubriques aussi.
La structure du FCB est :
Octet 0 : indique le numéro logique du fichier
Octet 1 : numéro du lecteur
Octet 2 : type du fichier : 0 pour un fichier à accès direct
Octet 3 : protection du fichier : même convention que pour les fichiers séquentiels.
Octets 4,5 : numéro de piste et de secteur du FCB

Octets 6,7 : nombre d'enregistrements par liste/secteurs
Octets 8,9 : nombre total d'enregistrements
Octets 10,11 : numéro enregistrement en cours d'exploitation
Octes 12 : longueur d'un enregistrement (en octets)
Octet 13 : nombre d'enregistrements par secteur
Octets 14,15 : n° piste et secteur de la première liste pistes/secteurs
Octets 16,17 : n° piste et secteur de la deuxième liste pistes/secteurs.
Octets 30,31 : n° piste et secteur de la 9e liste pistes/secteurs.

Le dernier buffer est le buffer entrée/sortie. Toutes les informations stockées ou à stocker sur la disquette transitent par ce buffer.

Voici deux routines qui permettent de lire un secteur de la disquette (son contenu est placé dans le Buffer E/S)
- d'écrire sur un secteur le contenu du Buffer E/S
le contenu du Buffer E/S
- Mode opératoire pour lire un secteur
- Autoriser l'entrée dans l'Overlay (interdire interruptions VIA)
Charger # C007 avec le n° du disque
Charger # C008 avec le numéro de face
(0 = face A, 1 = face B, pour les lecteurs simple tête toujours 0)
Charger # C009 avec le numéro de piste (0-40 pour les lecteurs simples, 0 81 pour les double-têtes)
Charger # C00A avec le numéro du secteur (1 à 17)
Appeler le sous-programme JSR # C24E
le contenu du secteur sélectionné est dans le buffer.
- Pour écrire un secteur :
Procéder de la même manière, mais remplir le buffer E/S avant d'appeler la sous routine qui est cette fois-ci JSR # C246...

A titre d'exemple d'application des notions évoquées ci-dessus voilà un programme qui permet de classer dans l'ordre alphabétique les fichiers constituant le catalogue.

```
>%B800,B8FD
B800 D8      CLD
B801 78      SEI
B802 A97F    LDA %#7F
B804 8D0E03  STA #030E
B807 A901    LDA %#01
B809 8DFA03  STA #03FA
B80C 8D07C0  STA #C007
```

```
B80F A914   LDA %#14
B811 8D09C0 STA #C009
B814 A902   LDA %#02
B816 8D0AC0 STA #C00A
B819 A900   LDA %#00
B81B 8D08C0 STA #C008
B81E 8550   STA #50
B820 8552   STA #52
B822 A950   LDA %#50
B824 8553   STA #53
B826 8555   STA #55
B828 A912   LDA %#12
B82A 8554   STA #54
B82C A94F   LDA %#4F
B82E 8551   STA #51
B830 E651   INC #51
B832 204EC2 JSR #C24E
B835 A000   LDY %#00
B837 B900FE LDA #FE00,Y
B83A 9150   STA (#50),Y
B83C C8     INY
B83D D0F8   BNE #B837
B83F A002   LDY %#02
B841 B150   LDA (#50),Y
B843 F00E   BEQ #B853
B845 300C   BMI #B853
B847 8D09C0 STA #C009
B84A C8     INY
B84B B150   LDA (#50),Y
B84D 8D0AC0 STA #C00A
B850 4C30B8 JMP #B830
B853 E651   INC #51
B855 A006   LDY %#06
B857 C8     INY
B858 B152   LDA (#52),Y
```

```
B85A D154   CMP (#54),Y
B85C F0F9   BEQ #B857
B85E 9011   BCC #B871
B860 A004   LDY z#04
B862 B152   LDA (#52),Y
B864 48     PHA
B865 B154   LDA (#54),Y
B867 9152   STA (#52),Y
B869 68     PLA
B86A 9154   STA (#54),Y
B86C C8     INY
B86D C016   CPY z#16
B86F D0F1   BNE #B862
B871 A554   LDA #54
B873 18     CLC
B874 6912   ADC z#12
B876 8554   STA #54
B878 C9FC   CMP z#FC
B87A D0D9   BNE #B855
B87C A900   LDA z#00
B87E 8554   STA #54
B880 E655   INC #55
B882 A555   LDA #55
B884 C551   CMP #51
B886 90CD   BCC #B855
B888 A553   LDA #53
B88A 8555   STA #55
B88C A552   LDA #52
B88E 18     CLC
B88F 6912   ADC z#12
B891 8552   STA #52
B893 8554   STA #54
B895 C9FC   CMP z#FC
B897 D0DB   BNE #B874
B899 A900   LDA z#00
B89B 8552   STA #52
```

```
B89D A912   LDA %#12
B89F 8554   STA #54
B8A1 E653   INC #53
B8A3 A553   LDA #53
B8A5 8555   STA #55
B8A7 C551   CMP #51
B8A9 90AA   BCC #B855
B8AB A950   LDA %#50
B8AD 8553   STA #53
B8AF A900   LDA %#00
B8B1 8552   STA #52
B8B3 20F2B8 JSR #B8F2
B8B6 A914   LDA %#14
B8B8 8D09C0 STA #C009
B8BB A902   LDA %#02
B8BD 8D0AC0 STA #C00A
B8C0 2046C2 JSR #C246
B8C3 E653   INC #53
B8C5 A553   LDA #53
B8C7 C551   CMP #51
B8C9 F01B   BEQ #B8E6
B8CB 20F2B8 JSR #B8F2
B8CE A000   LDY %#00
B8D0 B152   LDA (#52);Y
B8D2 8D09C0 STA #C009
B8D5 C8     INY
B8D6 B152   LDA (#52),Y
B8D8 8D0AC0 STA #C00A
B8DB 2046C2 JSR #C246
B8DE E653   INC #53
B8E0 A553   LDA #53
B8E2 C551   CMP #51
B8E4 D0E5   BNE #B8CB
B8E6 A900   LDA %#00
B8E8 8DFA03 STA #03FA
B8EB A9C0   LDA %#C0
```

```
B8ED 8D0E03 STA #030E
B8F0 58     CLI
B8F1 60     RTS
B8F2 A000   LDY %#00
B8F4 B152   LDA (#52),Y
B8F6 9900FE STA #FE00,Y
B8F9 C8     INY
B8FA D0F8   BNE #B8F4
B8FC 60     RTS
>J
```

CLASCAT.BIN

indique un chiffre hexadécimal
% indique adresse immédiat

Le programme est logé des adresses # B800 à # B900.

Le principe : le catalogue est lu secteur après secteur et transféré en RAM utilisateur, puis on effectue le classement, enfin on enregistre le catalogue ainsi modifié.

B801-B80B : préparation de l'entrée dans l'Overlay
B80C-B81D : préparation pour lire le premier secteur du catalogue (toujours piste = 14, secteur = 02)
B81E-B82F : initialisation des pointeurs dans le catalogue 50,51 : position de départ pour le transfert du buffer d'E/S en RAM utilisateur. Ici # 5000 : le catalogue sera transféré à partir de cette adresse.
52,53 : pointe le premier mot à comparer
54,55 : pointe le deuxième mot à comparer
B832 : lecture d'un secteur catalogue
B835-B83E : transfert du contenu du buffer E/S en RAM user.
B83F-B854 : recherche si lien de chaînage et lecture des secteurs suivants le cas échéant.
B855-B85F : comparaison de deux mots
B860-B870 : inversion des deux mots si le premier plus grand que le second
B871-B8AA : mise à jour des pointeurs (52,53) et (54,55) ; le tri employé est le tri à bulle

B8AB-B8BA : préparation du transfert inverse de # 5000 ▸ buffer entrée/sortie ; transfert
B8BB-B8C2 : écriture du premier secteur du nouveau catalogue
B8C3-B8E5 : recherche et écriture (le cas échéant) des secteurs suivants
B8E6-B8F1 : remise en condition normal, retour au basic... etc.
B8F2-B8FC : S/programme assure le transfert d'une page de la mémoire user ▸ la mémoire Overlay.

REPERTOIRE DES INSTRUCTIONS DU TDOS

! SAVE«(numéro de disque.)nom du fichier(.type)(,AD,AF)»

Cette instruction permet de sauvegarder des fichiers de type BASIC ou de type BINAIRE.

Numéro de disque : il s'agit de l'unité dans laquelle se trouve placée la disquette sur laquelle le programme en question sera sauvegardé. Par défaut, ce numéro est 1 (cas des possesseurs d'un seul lecteur). Pour les autres, l'unité concernée devra avoir été préalablement montée (par ! MOUNT«numéro de disque.»). Ne pas oublier le Point . après le numéro.

Nom du fichier : il s'agit du nom sous lequel sera sauvegardé le fichier et qu'il portera dans le catalogue. Ce nom doit obéir à la syntaxe habituelle ; en particulier, il ne doit pas comporter d'espace. Ce nom sera dans tous les cas tronqué à 8 caractères.

(.type) : cette indication n'est pas obligatoire pour un fichier BASIC, c'est-à-dire pour un programme BASIC. Elle est par contre indispensable pour un fichier BINAIRE, c'est-à- dire pour une zone mémoire de l'unité centrale. Celle-ci peut être une suite d'octets ou bien un programme en langage machine. Par commodité, on peut utiliser les types : BIN,DAT,TXT dans le premier cas, le type : CMD dans le second. Seuls les fichiers binaires de type : BIN ou CMD sont exécutables.

(,AD,AF) : Dans le cas d'un fichier BINAIRE, il faut préciser dans la commande l'adresse à laquelle débute la zone mémoire que l'on veut sauvegarder. Cette adresse est ici symbolisée par AD, cette quantité pouvant être un nombre (exprime en décimal ou en hexadécimal) ou bien encore une variable. De même, AF symbolise l'adresse de fin de zone mémoire à sauvegarder.

L'ensemble des quantités situées entre guillemets peut être remplacée par une chaîne.

EXEMPLE 1
A$=«FILTRE»
A$=«2.»+A$
! SAVEA$
Cette suite d'instructions permet de sauvegarder sur le lecteur numéro 2 le programme BASIC en mémoire sous le nom : FILTRE.

EXEMPLE 2
! SAVE«CARACTER.BIN,#B400,#B800»
Cette instruction sauve sur disquette (le premier lecteur par défaut) le jeu de caractères standard de l'ATMOS.

REMARQUE : Cette instruction ne peut pas être utilisée pour les fichiers à accès séquentiel ou à accès direct (créés par ! CREATE), conservés dans le catalogue sous le type : DAT (data).

! LOAD«(numéro de disque.)nom du fichier(.type)(,AD)»
numéro de disque : voir instruction ! SAVE
nom du fichier : voir instruction ! SAVE

(.type) : pour un programme BASIC, le type .BAS est facultatif. Pour un fichier BINAIRE (type BIN,TXT,SCR,DAT ou CMD), il est obligatoire. Dans ce dernier cas, il y a deux possibilités : soit le fichier sera chargé à la place qu'il occupait au moment de son ! SAVE (cas d'un fichier SCR, qui se replace automatiquement en mémoire vive écran, et qui a été sauvegardé par ! LSCR ou ! HSCR(, auquel cas l'indication suivante (,AD) n'est pas obligatoire, soit on désire le charger à un endroit différent, et la suite de l'instruction : (,AD) doit être fournie.

!«(numéro de disque.)nom du fichier(.type)»
Cette instruction permet de charger un programme et de le lancer.
(numéro de disque.) : Numéro de l'unité de disque concernée (1 à 4);sa valeur est 1 par défaut.
nom du fichier:voir instruction ! SAVE.
(.type) : L'indication .BAS est facultative pour un programme BASIC. Elle est par contre obligatoire pour un programme en langage machine (type .BIN ou .CMD) exécutable.

EXEMPLE 1
! «FILTRE»
Chargement et lancement automatique du programme BASIC FILTRE

! SEARCH «(numéro disque.)**nom fichier.type**»

Cette instruction permet de s'assurer de l'existence d'un fichier. (.type) est obligatoire même pour un fichier BASIC. Cette instruction est particulière dans la mesure où sa seule conséquence est de déclencher l'émission d'un message d'erreur.

		mode immédiat	mode programme	mode programme + ! ERSET
! SEARCH ↓	le fichier existe	"EXISTING FILE"	"EXISTING FILE"	— ⧫ 1 en # 489
	il n'existe pas	"FILE NOT FOUND"	"FILE NOT FOUND"	— ⧫ 7 en # 489
puis →		Ready	Break Ready	le programme se poursuit

! LOCK«(numéro disque.) nom fichier.type»
! UNLOCK «(numéro disque.) nom fichier.type»

Ces 2 instructions permettent respectivement de verrouiller (déverrouiller) un fichier sur disquette. (Indications : L et U dans le catalogue). Un fichier verrouillé ne peut :
- être effacé par ! DEL
- être écrasé par ! SAVE
- être étendu par ! WRITE

! DEL «(numéro disque.)nom fichier.type»

Cette instruction permet de détruire (et donc de faire disparaître du catalogue) un fichier non verrouillé et ainsi de récupérer sa place sur la disquette.

! RENAME«(numéro disque.)NOMA(.type)=NOMB(.type)»

Cette instruction permet de changer le nom d'un fichier présent sur la disquette (ancien nom = NOMA, nouveau nom =NOMB). Le type reste cependant le même. Seul le DOS, dont le nom a été fourni au cours de l'instruction ! MASTER, ne peut être modifié par cette instruction.

! DNAME«(numéro disque.)nom volume»

Cette instruction permet de modifier le nom qui a été donné à la disquette lors de l'instruction ! INIT.

! «FORMAT»

Il s'agit-là d'un utilitaire BASIC, permettant de formatter une disquette, c'est-à-dire de la découper en secteurs et en piste. Cette opération est conseillée (ainsi que ! INIT) pour « régénérer » une disquette déjà utilisée. Ce programme, situé sur une disquette d'utilitaire ne lui fait courir aucun risque (contrairement aux apparences...), tant que l'indication « changez de disquette » n'apparaît pas.

! INIT «(numéro de disque.)nom volume»

Cette commande est la suite logique de ! «FORMAT», mais c'est le TDOS situé dans la RAM en Overlay qui l'effectue. Celui-ci devra au préalable avoir été chargé. C'est la disquette située dans le lecteur qui sera alors initialisée ; atttention, car cette opération détruit tous les fichiers présents sur la disquette. Son rôle est d'initialiser le catalogue.

! MASTER «(numéro de disque.)nom du DOS»

Cette instruction permet de transférer le DOS (préalablement chargé) présent en RAM, sur une disquette fraîchement formattée et initialisée. Elle doit prendre place immédiatement après ! INIT. Une telle disquette devient alors « maître », c'est-à-dire autonome et capable elle-même de charger le TDOS en RAM, après le BOOT.

! CAT «(numéro de disque.)» et ! LCAT «(numéro disque.)»

Ces instructions permettent de lister le catalogue sur l'écran (! CAT) ou sur l'imprimante (! LCAT). Le catalogue est l'ensemble des noms des fichiers présents sur la disquette et leurs caractéristiques :

L ou U	nom fichier	type	S ou D	xxxSECTORS
protégés ou non			accès séquentiel ou direct	occupation sur la disquette

Un appui sur la touche d'espace permet d'interrompre le listing ; à chaque appui suivant, un nouveau fichier sera dévoilé. Poursuivre ainsi qu'à l'apparition de Ready, un «Return» interrompt le catalogue.

! MOUNT «(numéro disque.)» et ! DEMOUNT«(numéro disque.)»

Ces 2 instructions, aux effets opposés, permettent de signaler au TDOS l'existence (ou la disparition) de lecteurs en ligne, distincts du premier. Il est conseillé pour les utilisateurs possédant plusieurs lecteurs de placer l'instruction ! MOUNT dans le programme ! START. L'oubli de cette commande est signalé par le message d'erreur : «DRIVE NOT IN LINE».

! COPY «(numéro disque.)nom fichier A (.type) = (numéro disque.)nom fichier B (.type)»

Cette instruction permet de dupliquer un fichier, soit sur une même disquette (il faut alors changer de nom ; dans le cas contraire, il y a évidemment émission du message d'erreur «EXISTING FILE !»), soit sur une autre disquette. Les caractéristiques à gauche du signe = sont celles de la source (émetteur), celles à droite sont celles du récepteur. Voir le texte pour les fichiers en langage machine.

! CUT N

Cette instruction permet (pour un programme BASIC situé en mémoire centrale) de supprimer toutes les lignes à partir de celle de numéro N. Toutes les variables sont conservées au cours de cette opération.

! MERGE «(numéro de disque.) nom du programme»

Cette instruction permet de chaîner un programme sur disquette à un programme existant en mémoire centrale, dans la mesure où toutes les lignes de celui-ci ont des numéros inférieurs à celles de l'autre. Couplée avec ! CUTN, cette instruction puissante permet de faire tourner des programmes BASIC distincts simultanément, ou bien dont l'encombrement serait supérieur à la place disponible en mémoire centrale. Dans les cas délicats, l'utilisateur aura intérêt à employer l'un des nombreux programmes de rénumération que l'on trouve dans la littérature.

! «TKD»

Il s'agit-là d'un utilitaire qui permet de transférer en bit à bit des informations d'une cassette sur la disquette. Les programmes concernés

ne seront par la suite à modifier que s'ils font appel eux-mêmes aux lecteurs de cassette. Dans les cas les plus simples :

CSAVE ➧ ! SAVE
CLOAD ➧ ! LOAD
STORE ➧ ! MSAVE
RECALL ➧ ! MLOAD
! LSCR «(numéro disque.)nom du fichier»

Cette instruction permet de sauvegarder un écran texte, tel qu'il apparaît. Elle est l'équivalent de :
! SAVE«(numéro disque.)nom du fichier.SCR,#BB80,#BFDF»

Pour quelle ne soit pas elle-même sur l'écran, elle peut être, soit incluse dans un programme, soit tapée de façon invisible (encre de la couleur du papier) par ESCAPE + code de couleur d'encre).

! HSCR«(numéro disque.)nom fichier»

Permet de sauvegarder sur disquette un écran haute résolution. Elle est l'équivalent de :
! SAVE«(numéro disque.)nom fichier.SCR,#A000,#BFDF»

Tout comme son homologue ! LSCR, ces écrans peuvent être récupérés de la manière suivante :
- se placer dans le mode adéquat (TEXT) ou (HIRES)
- appeler l'écran par :
! LOAD «(numéro de disque.) nom fichier»

! MSAVE «(numéro disque.)nom fichier = nom tableau»

Permet de sauvegarder un tableau (ou une matrice). Le nom fichier est son nom tel qu'il figurera dans le catalogue, nom tableau est le nom qu'il porte dans l'unité centrale. Ainsi, un tableau A défini par A (10,10) sera sauvegardé par :
! MSAVE«TABLEAU=A»

Les tableaux d'entiers (A%) et de chaînes (A$) seront sauvegardés de la même manière.

! MLOAD «(numéro disque.)nom tableau»

Permet de récupérer un tableau préalablement stocké sur disquette. Le tableau récepteur devra avoir été dimensionné ; les dimensions seront de préférence identiques à celles du tableau émetteur.

! CREATE «(numéro de disque.)nom fichier, S = NLU»

Cette instruction crée et ouvre un fichier à accès séquentiel (S) portant le nom : nom fichier sur la disquette (de type : .DAT) et le numéro logique NLU vis à vis du TDOS. Dorénavant, et tant qu'il n'y a pas eu de ! CLOSE, toutes les opérations sur ce fichier se feront par l'intermédiaire de ce numéro logique.
(1≤NLU≤255). Ainsi, NLU résume

l'ensemble des 3 paramètres { numéro disque, nom fichier, séquentiel

! OPEN «(numéro disque.)nom fichier = NLU»

Cette instruction ouvre (elle permet donc de lire et d'y écrire) le fichier de nom : nom fichier, en lui attribuant le numéro logique NLU. Seuls 16 fichiers peuvent être ouverts simultanément.

! CLOSE NLU

Cette instruction ferme le fichier de numéro logique NLU. A partir de là, toute opération sur ce fichier est impossible, jusqu'au prochain ! OPEN. Il est vivement recommandé de fermer tout fichier dont on n'a plus besoin, en particulier avant d'éteindre l'ordinateur. ! DEL sur un fichier non fermé peut avoir des conséquences imprévisibles ! CLOSE 0 ferme la totalité des fichiers en cours.

! WRITE NLU'< liste de variables>ou
! WRITE NLU : REM<liste de variables> [pour ORIC 1]

Cette commande permet d'écrire dans le fichier NLU à l'enregistrement courant la valeur des variables dont le nom est situé entre les ‹› ; celles-ci y sont séparées par des virgules. Chaque variable correspond alors à une rubrique ; la liste des variables à un enregistrement. Une variable numérique réelle occupe 17 octets, entière, 8 octets, de chaîne, sa longueur + 2 octets. Une variable ne peut être un élément de tableau. Une rubrique peut être écrasée, ceci sans bousculer le fichier : la première écriture d'une rubrique impose sa longueur future (et définitive). Cette instruction est inhibée par ! WL et validée par ! WUL.

!TAKE NLU' ‹ liste de variables ›

Syntaxe identique à ! WRITE.

Une variable chaîne doit cependant avoir vu préalablement son nom créé. Ainsi si A$ n'a jamais été utilisé auparavant, ! TAKE NLU'A$ doit être précédé de A$ = «» ; dans le cas contraire, le message d'erreur « MISSING VARIABLE » est affiché.

! REWIND NLU

Cette instruction permet de positionner le pointeur de rubrique au début d'un fichier à accès séquentiel. Ainsi, la première rubrique est prête à être lue ou écrasée.

! REWIND ∅ repositionne la totalité des fichiers ouverts.

! APND NLU

Cette instruction positionne le pointeur de rubriques en fin de fichier, à la première rubrique non écrite. ! APND 0 effectue cette opération sur la totalité des fichiers ouverts.

! WHERE NLU

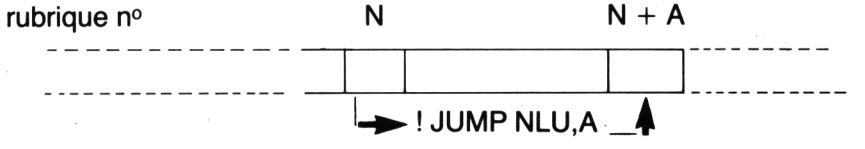
Cette instruction n'a aucune conséquence directe. Cependant, après son exécution, on trouve en # 48 D la position du pointeur de rubrique et en # 48 F le nombre total de rubriques du fichier plus un. Ainsi, pour un fichier de 17 rubriques, dont on vient de lire la quatrième :

```
! WHERE NLU
PRINT PEEK (# 48 D), PEEK (#48 F)
```

Conduit à : 5 18

! JUMP NLU, A

Permet d'incrémenter de A le pointeur de rubriques.



A = 0 est interdit, ainsi que ‹ 0. Cependant, si une telle éventualité peut se produire accidentellement, elle peut être neutralisée par ! ERSET.

! ERSET et ! EROFF ; ! ERR GO TO N

Ces deux premières instructions permettent respectivement d'empêcher et d'autoriser l'émission de messages d'erreur, et de poursuivre (ou d'arrêter) l'exécution d'un programme le cas échéant. Une erreur masquée par ! ERSET doit être traitée : l'instruction suivant celle qui peut engendrer l'erreur est alors la troisième. Dès son utilisation, l'emplacement mémoire # 489 contient le code corresondant à l'erreur.

Ces instructions ne permettent de traiter que les messages d'erreur du TDOS, et non ceux du BASIC.

! WL NLU et ! WUL NLU

Permettent respectivement d'empêcher et d'autoriser l'écriture dans un fichier ouvert. Cependant, elles ne peuvent s'opposer à un ! DEL, contre lequel la seule protection est ! LOCK.

! CREATE «(numéro disque.)nom fichier, D = NLU,LR,NE»

Création et ouverture d'un fichier à accès direct, de numéro logique NLU. LR est la largeur d'un enregistrement compté en octets (1 réel = 17 octets, 1 entier = 8 octets, 1 chaîne = sa longueur + 2), NE le nombre d'enregistrements prévus. Ce dernier paramètre n'est cependant pas limitatif : un fichier à accès direct peut être prolongé par un fichier à accès séquentiel.

! WRITE NLU,E' < liste de variables > ou
! WRITE NLU,E: REM < liste de variables > [pour ORIC 1]

Permet d'écrire à l'enregistrement E ($E \leqslant NE + 1$) la valeur des variables situées entre les ‹›, et séparées par des virgules. Un enregistrement est toujours écrit (ou écrasé) dans sa totalité. La partition des rubriques n'est pas rigide ; seule compte la longueur totale de l'enregistrement (en octets) telle qu'elle a été définie par ! CREATE.

! TAKE NLU,E' < liste des variables > ou
! TAKE NLU,E: REM < liste des variables > [pour ORIC 1]

Permet de lire dans un fichier à accès direct, à l'emplacement E, la valeur des variables dont la liste est donnée entre les crochets ‹›. Chaque enregistrement est lu à partir du début.

! « BKP »

Il s'agit d'un utilitaire, permettant de copier intégralement une disquette sur une autre. C'est éventuellement la seule manière de copier un programme en langage machine se logeant en page 400.

TABLE DES MESSAGES D'ERREUR

CODE	MESSAGE
1	EXISTING FILE
2	DRIVE NOT IN LINE
3	PROGRAM TOO LARGE
4	FILE TYPE MISMATCH
5	DISK FULL
6	I/O ERROR
7	FILE NOT FOUND
8	WRITE PROTECTED
9	RANGE ERROR
10	SYSTEM ERROR
11	SYNTAX ERROR
12	TOO MANY OPENED FILES
13	FREE MEMORY UNDERFLOW
14	MISSING VARIABLE
15	END OF FILE
16	OUT OF DATA

REPERTOIRE DES INSTRUCTIONS

Abreviations:
NL: numéro de lecteur de disquette de 1 a 4
NFICH:nom du fichier pris au sens large
TYP: type du fichier;ARY,BAS,BIN,CMD, DAT,SCR,SYS,TXT
AD:adresse de debut du bloc
AF:adresse de fin du bloc
NLU:numéro logique d'un fichier
(): argument optionnel sous conditions
S'il est omis le dos choisit une valeur par defaut.Sauf indications contraires les valeurs par defaut sont:
-1 pour NL
-BAS pour TYP

Liste des instructions du TDOS

A Opérations sur les disquettes

```
ºINIT"(NL.) NOM DISQUE"
ºMASTER"(NL.) NOM DU TDOS DANS LE CAT"
ºDNAME"(NL.) NOM DISQUE"
ºCAT"(NL.)"
ºLCAT"(NL.)"
ºMOUNT"NL."
ºDEMOUNT"NL."
ºCOPY"(NLE.)NFICHA(.TYP)=(NLR.)NFICHB(.
TYP)
   NLE=numéro lecteur émétteur
   NFICHA=nom du fichier à copier
   NLR=numéro du lecteur recepteur;NLR
peut être egal à NLE
   NFICHB=nom du fichier crée
```

B Opérations sur tous les fichiers

```
ºSAVE"(NL.) NFICH (.TYP)(,AD,AF)"
ºLOAD"(NL.) NFICH (.TYP)(,AD)"
º"(NL.) NFICH (.TYP)(,AD)"
ºSEARCH"(NL.) NFICH.TYP"
ºLOCK"(NL.) NIFICH.TYP"
ºUNLOCK"(NL.) NFICH.TYP"
ºDEL"(NL.) NFICH.TYP"
ºMERGE"(NL.) NFICH"
ºRENAME"(NL.)NFICHA(.TYP)=NFICHB(.TYPB)
   NFICHA=nom du fichier à renommer
   NFICHB=nouveau nom
   TYPB=nécessaire si différent de TYP
   TYP ne doit pas être SYS
ºCUT N
   N=numéro de ligne
```

C Traitement des erreurs

```
ºERSET
ºEROFF
ºERR GOTO N
   N=numéro de ligne BASIC
```

D Traitement des fichiers de données

ºWL NLU
ºWUL NLU
ºOPEN"(NL.)NFICH=NLU"
ºCLOSE NLU
Si NLU=0 tous les fichiers ouverts sont fermés

E Traitement des fichiers à accés séquentiel

ºCREATE"(NL.) NOMFICH,S=NLU"
ºWRITE NLU'<liste de variables>
ºWRITE NLU:REM<liste de variables>
Seule la 2ieme syntaxe est valable pour l'ORIC 1
ºTAKE NLU'<liste de variables>
ºTAKE NLU:REM<liste de variables>
Seule la 2iéme syntaxe est valable pour l'ORIC 1
ºREWIND NLU
ºAPND NLU
ºWHERE NLU
ºJUMP NLU,NI
NI=nombre d'incrémentations du pointeur

F Traitement des fichiers à accés direct

ºCREATE"(NL.)NFICH,D=NLU,LR,NE"
LR=longueur d'un enregistrement
NE=nombre d'enregistrements reservés à l'avance
ºWRITE NLU,E'<liste de variables>
ºWRITE NLU;E:REM<liste de variables>
Seule la 2iéme syntaxe est valable pour l'ORIC 1
ºTAKE NLU,E'<liste de variables>
ºTAKE NLU,E:REM<liste de variables>
Seule la 2iéme syntaxe est valable pour l'ORIC 1

G Opérations sur les matrices

ºMSAVE"(NL.) NFICH=NOMAT"
ºMLOAD"(NL.) NFICH=NOMAT"
NOMAT=nom de la matrice en mémoire centrale

H Sauvegardes d'écran

ºLSCR"(NL.)NFICH"
ºHSCR"(NL.)NFICH"

I Utilitaires divers

º"FORMAT"
º"TKD"
º"BKP"
º"COPY1"

Imprimerie DUPLI-OFFSET - TOULON - Tél. 27.15.69

Le **T.DOS** ET SES FICHIERS pour **ORIC 1 & ATMOS**

P.M. BEAUFILS

C. ARNAUD

La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les "copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective", et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, "toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit, ou ayants cause, est illicite" (alinéa 1er de l'article 40).

Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles 425 et suivants du Code Pénal.

© Dépôt légal septembre 1984
Technologie. Recherche & Applications Nouvelles
53, Impasse Blériot - 83130 LA GARDE

Pierre-Marc BEAUFILS, agrégé de physique, enseigne l'informatique et l'électronique dans les classes supérieures depuis une dizaine d'années. Il a d'ailleurs écrit de nombreux ouvrages sur ce sujet.

Christian ARNAUD prépare la maîtrise en informatique et est mordu comme beaucoup, du virus de la micro.